(4)

RADC-TR-89-317
Final Technical Report
January 1990

AD-A220 116

SOFTWARE QUALITY MEASUREMENT METHODOLOGY ENHANCEMENTS STUDY RESULTS

Rochester Institute of Technology

Jeffrey A. Lasky, Alan R. Kaminsky, Wade Boaz

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-89-317 has been reviewed and is approved for publication.

APPROVED: Roger B. Panara

ROGER B. PANARA
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Igor G. Plonisch

IGOR G. PLONISCH
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific doucment require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-317
6a. NAME OF PERFORMING ORGANIZATION Rochester Institute of Technology 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)
6c. ADDRESS (City, State, and ZIP Code) One Lomb Memorial Drive P O Box 9887 Rochester NY 14623-0887	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) COEE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-81-C-0193
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO: 62702F; PROJECT NO: 5581; TASK NO: 20; WORK UNIT ACCESSION NO: P5

11. TITLE (Include Security Classification)

SOFTWARE QUALITY MEASUREMENT METHODOLOGY ENHANCEMENTS STUDY RESULTS

12. PERSONAL AUTHOR(S)

Jeffrey A. Lasky, Alan R. Kaminsky, Wade Boaz

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Dec 87 TO Mar 89	January 1990	150

16. SUPPLEMENTARY NOTATION

N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software quality, Software quality metrics
12	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The four (4) tasks and their results performed under this study are: 1. Perform a critical analysis of the RADC Software Quality Measurement Methodology. Methodology is complex. The specification process should be modified. 2. Determine if AFSC Management and Quality Indicators can be integrated into the RADC methodology. Three (3) of the Quality Indicators can be integrated. 3. Review completeness of Methodology Traceability through software life cycle phases. In general, traceability is present. 4. Create a cross reference guide between Methodology life cycle phase metrics and applicable DOD-STD-2167A DIDs paragraphs. A cross reference matrix was developed; however, many problems exist and recommendations are made to alleviate the problems.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Roger B. Panara	22b. TELEPHONE (Include Area Code) (315) 330-3655 22c. OFFICE SYMBOL RADC (COEE)

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

TABLE OF CONTENTS

V. STUDY of METRIC QUESTION TRACEABILITY

VI. STUDY of the RELATIONSHIP BETWEEN METRIC DATA COLLECTION and 2167A

APPENDIX A

CROSS REFERENCE GUIDE

FIGURES

TABLES

SECTION I

EXECUTIVE SUMMARY

1. INTRODUCTION

This technical report presents results of the Software Quality Measurement Methodology Enhancement Study. This work was performed for Rome Air Development Center (RADC) by Rochester Institute of Technology (RIT) under contract no. F30602-81-C-0193.

2. BACKGROUND

The focus of this 12 month effort was to continue work on the software quality measurement methodology which has been under development by RADC since 1976. Most of the current RADC sponsored software quality efforts, including this study, are extensions or enhancements of the methodology as set forth in a three volume set of guidebooks developed by Boeing Aerospace Company and published in early 1985 [1]. Volumes II and III of the guidebooks provide the software acquisition manager with guidelines for the quantitative specification and evaluation of software quality. The software life-cycle phases and terminology used in the guidebooks was consistent with the December, 1983 version of DOD-STD-SDS (DOD-STD-2167).

After the publication of the guidebooks, RADC sponsored three guidebook validation projects. These projects were undertaken in the 1985-1987 time period. The purpose of these projects was to assess the feasibility and utility of transitioning the software quality measurement methodology to the software acquisition environment [2,3,4]. Each of these validation projects generated numerous recommendations for modifications to the methodology and to the guidebooks.

In 1986 and 1987, Air Force Systems Command (AFSC) developed a measurement-based approach to help manage the process of software development [5,6]. These process metrics were designed for high-level process management in contrast to RADC's more detailed level product management metrics.

In early 1988, DOD issued the A revision of DOD-STD-2167 [7]. Of particular relevance to the study reported here, most of the software quality requirements which were contained in DOD-STD-2167 were not carried forward into DOD-STD-2167A. Instead, a separate software quality standard, DOD-STD-2168, was also issued in 1988 [8].

Throughout 1988, several meetings of the Software Quality Methodology Working Group were held. The members of the working group were primarily RADC contractors who had ongoing efforts in the software quality measurement program. The general purpose of the these meetings was to facilitate exchange of technical information and to provide a mechanism for coordination of related efforts. A specific focus of the Working Group was to review the large set of issues and recommendations generated by prior related work and then to resolve these items by defining a baseline version of the methodology. The results of the Working Group's meetings are found in [9]. Organizations represented in the Working Group included Computer Science Innovations, Inc. (CSI), the Data and Analysis Center for Software (DACS), Dynamics Research Corporation (DRC), Illinois Institute of Technology Research Institute (IITRI), RADC, RIT, Scientific Applications International Corporation (SAIC), Software Productivity Solutions, Inc. (SPS), and US Army Communications-Electronics Command (CECOM).

3.2. Relationship to other related efforts and activities

As we began our work, it soon became apparent that some of our efforts would be subsumed within larger efforts, while other assigned tasks would be precluded by other factors. This section discusses the impact of these factors on our initial work scope.

Our participation in the Software Quality Methodology Working Group, noted above, impacted tasks 1(a) and 1(b). Since the Working Group assumed a broad perspective in reviewing the methodology, and since the authors of the three validation studies were members of the Working Group, we believed it would be most efficient for us to provide our evaluations in the context of the Working Groups activities. Accordingly, this report includes a study on an area not considered by the Working Group, namely the Guidebook Volume II processes which implement the software quality specification component of the methodology. Also included is a first description of a significantly different and alternative view of the methodology. Our recommendations, related to the other issues in 1(a) and 1(b),are included in [9].

The issuance of DOD-STD-2168, Defense System Software Quality Program, impacted tasks 2(a) and 2(b). The purpose of this standard "is to establish requirements for a software quality program to be applied during the acquisition, development, and support of software ststems" [8, p.1]. DOD-STD-2167, issued prior to DOD-STD-2168, reflected software quality concerns. These quality concerns were either largely eliminated or substantially subordinated when DOD-STD-2167A was issued, since work on DOD-STD-2168 was nearly complete. Thus, it was no longer necessary to pursue tasks 2(a) and 2(b). In addition, the degree of generality, which we believe was ill-advised, incorporated in the wording of DOD-STD-2167A, would have made problematical the task of writing specifc, unambiguous metric questions intended to be consistent with DOD-STD-2167A.

3.3. Revised work scope and deliverables

Based on the discussion in section 3.2, our activities were focused on the following:

1. Enhance the software quality measurement methodology by:

   (a) presenting a new view of the methodology which can be considered as either complementary to or as an alternative to the classical view of the methodology. This view is outlined in Section II, Elements of a Software Quality Compliance Model.

   (b) examining the Guidebook Volume II processes which implement the software quality specification component of the methodology. Our recommendations for modifying the Volume II processes are contained in Section III, Study of the Software Quality Specification Process.

   (c) determining to what extent and how the AFSC software management and software quality indicators should be integrated into the methodology . Our analysis and approach is contained in Section IV, Study of AFSC Management Indicators.

2. Insure that traceability of metric questions is maintained under DOD-STD-2167A by reviewing the traceability of the methodology's metric questions across the DOD-STD-2167A lifecycle. Our analysis is contained in Section V, Study of Metric Question Traceability.

3. Improve the ease of use and cost effectiveness of implementing the methodology by developing a cross-reference guide which indicates the DOD-STD-2167A DID information source, if any, for each of the metric questions. Our cross-reference guide is contained in Section VI, Study of the Relationship Between Metric Data Collection and 2167A.

4. OVERVIEW OF CONTRACT RESULTS

4.1. Organization of the Report

Due to the wide scope of the investigations conducted under this contract, we decided to organize this report as a series of separate studies, each with its own introduction and set of references. None of the studies makes reference to any of the other studies. However, the first two studies discussed below in sections 4.2 and 4.3 are in friendly opposition; this state of affairs requires a brief explanation. The purpose of our investigations was to make recommendations concerning the enhancement of the methodology as described in the guidebooks. We followed this direction in our Study of the Software Quality Specification Process (and in all of the following studies as well). However, based on our initial and continuing review of the methodology, and our review of related work being conducted in parallel, we concluded that interjection of a new perspective was timely and potentially valuable. Thus, the reader should consider our too brief remarks in Section II about a quality compliance model separate from the remainder of the report.

Following are summaries of the results of our studies

4.2. Elements of a Software Quality Compliance Model

The RADC software quality measurement methodology is based on a 3 level hierarchial model. The top level is a set of customer-oriented *software quality factors*. The second level is a larger set of defining attributes for the software quality factors. These are termed *software quality criteria* and reflect technical considerations of good software engineering and development practice. The third level is a still larger set of *software quality metrics* which are believed to be valid component measures of the software quality criteria. The software quality metrics in turn are defined by lower level metric elements.

Guidebook Volume II provides a methodology for specifying software quality goals. The end result of the specification process is a set of quantitative goals for each of the top level software quality factors. Guidebook Volume III provides a methodology for evaluating achieved levels of software quality for intermediate and final software products. The evaluation is based on answers to the low level metric element questions. Most of these questions generate values of 1 if Yes and 0 if No. These values are then aggregated to generate metric scores which are in turn aggregated to generate criteria scores. The criteria scores are then aggregated to generate the factor scores which are compared to the factor goals. We have termed this approach the *Software Quality Aggregate Scoring Model.*

From the academic point of view, we are concerned that the compound effects of score averaging, weighting and aggregation will hinder efforts to validate this model. Also, we believe parts of the methodology are unnecessarily complex relative to the goal of improving the quality of delivered software. Our research contractor colleagues agree with this assessment. Unfortunately, demonstrated validity will be a critical success factor in the eventual acceptance of the methodology by the contracting community. Our concerns over this matter have lead us to propose a more simple and focused model we have named the *Software Quality Compliance Model.*

The key idea for the compliance model was suggested by work done at Hughes Aircraft Company by Deutsch and Willis [11]. They asserted that the Guidebook Volume III metric questions should be rephrased in terms of testable requirements. With this perspective in mind, we conceptualized a two-level model by eliminating the middle level criteria level from the RADC model. The top level quantitative factor goals are replaced by a set of required (mandatory) software features. The extent of the features required for a given quality factor is dependent on the desired quality level for that factor. Assessment of achieved quality is a matter of determining the percentage of required software features actually implemented. Hence, the term compliance model. We also see a real

need to provide a technical assessment of implementation (source code) quality. We envision this being accomplished automatically by a quality-oriented source code analyzer.

4.3. Study of the Software Quality Specification Process

Guidebook Volume II provides a methodology for specifying software quality goals. We believe the specification methodology should be simplified and then recast within the framework of AFSC Pamphelt 800-45, *Software Risk Management.*

A summary of our analysis is as follows. The current software quality specification process is too detailed and is not validated. It is too detailed because it requires specifying separate software quality factor goals for each identified software function, whereas we believe it is more appropriate to specify overall system quality factor goals or, at most, CSCI-level quality factor goals. It is also too detailed in that the system acquisition manager must devote significant amounts of time to determine what are meant to be broad quality goals; the amount of work required is out of proportion with the scope and accuracy of the results. The current process is also unvalidated. In particular, the negative factor interrelationships and incremental quality factor costs are based only on opinions and experience, not on actual real-world project data.

A summary of our recommendations is as follows. The software quality specification process is simplified by specifying quality factor goals at the system level, or at most the CSCI level. The process is also simplified by eliminating the tailoring of metric and criterion weights; all metric questions for a criterion and all criteria for a factor are weighted equally. The feasibility of meeting all the quality factor goals is studied as part of a risk analysis, which we recommend be conducted in accordance with AFSC Pamphlet 800-45, *Software Risk Management.* Software quality risk has two components: technical risk and cost risk. We suggest that the technical risk assessment be based on a simplified analysis of negative factor interrelationships. We also suggest that the cost risk assessment be based on an analysis of incremental costs derived from the COCOMO software cost model, which is a validated model based on real-world data. The software quality risk becomes one part of the overall risk analysis and may be handled in various ways, including assuming the risk and making no changes in the quality factor goals, or avoiding the risk by changing some of the quality factor goals.

4.4. Study of AFSC Management Indicators

The Air Force Systems Command (AFSC) has published two pamphlets in the area of software quality management,

- *Software Management Indicators,* AFSC Pamphlet 800-43.
- *Software Quality Indicators,* AFSC Pamphlet 800-14.

This study reports on the relationship between the Indicators and the Methodology and describes how and to what extent the Indicators can be incorporated into the Methodology.

A summary of our recommended approach is as follows. All 6 Software Management Indicators and 4 of the Software Quality Indicators are process-oriented and should not be directly incorporated into the product-oriented Methodology. These 10 Indicators are grouped to form the "Software Process Indicators." The remaining 3 Software Quality Indicators are product-oriented. These are grouped to form the "Software Product Indicators" and are incorporated into the Methodology. During development of a software project, the Software Process Indicators are calculated every month based on a small amount of collected process data. At the end of each development phase, a much larger amount of process and product data is collected. The process

data is used to calculate the Software Process Indicators as usual, and the product data is used to answer the Methodology metric questions. The Methodology metric questions in turn are used to calculate the Software Product Indicators and the Methodology metric, criteria, and factor scores. With this approach, the acquisition manager obtains both an overall view of the project's quality (from the Indicators) and a detailed view of each individual software document's quality (from the Methodology).

4.5. Study of Metric Question Traceability

Appendix A of Volume III contains a cross reference listing showing all the metric worksheets on which each metric question appears. For example, metric question CP.1(2), part of the Completeness criterion, is asked during the System/Software Requirements Analysis phase on Metric Worksheet 0; during the Software Requirements Analysis phase on Metric Worksheet 1; during the Preliminary Design phase on Metric Worksheet 2; during the Detailed Design phase on Metric Worksheets 3A and 3B; and during the Coding and Unit Testing phase on Metric Worksheets 4A and 4B. This question is not asked during the CSC Integration and Testing, the CSCI-Level Testing, or the System Integration and Testing phases.

It can be seen that metric question CP.1(2) is asked repeatedly during each sequential development phase. Thus, the quality concern this question addresses is tracked from phase to phase to ensure that it is addressed throughout the development process. We say that *traceability* is provided for this quality concern across the development process.

Studying Appendix A, one quickly discovers that there seem to be traceability gaps in certain metric questions. For example, the AM.1(1) metric question is asked during the System/Software Requirements Analysis phase on Metric Worksheet 0 and during the Software Requirements Analysis phase on Metric Worksheet 1. Subsequently, that question appears on no other worksheet and is not asked during the Preliminary Design, Detailed Design, Coding and Unit Testing, and CSC Integration and Testing phases. After this gap, the question reappears during the CSCI-Level Testing phase on Metric Worksheet 1 and the System Integration and Testing phase on Metric Worksheet 0. Many other examples are apparent in Appendix A.

This study addresses the following questions:

- Is it true that because of these traceability gaps, important software quality concerns will not be addressed during certain development phases?
- Should new metric questions be written to fill the gaps?
- If so, what are the metric questions that should be added?

We studied each metric question gap and concluded that in all cases there are valid reasons for the gap's existence. Traceability of software quality concerns is not lost due to these gaps. No new metric questions need to be added.

4.6. Study of the Relationship Between Metric Data Collection and 2167A

The primary goal of this study was to develop a cross-reference guide which indicates the DOD-STD-2167A DID information source, if any, for each of the metric questions. The intent of the cross-reference guide is to improve the ease of use and cost effectiveness of implementing the methodology by reducing the time required to locate information required to answer the metric questions. A secondary goal was to consider the relationship between metric data collection and DOD-STD-2167A.

In general, we found the process of identifying DOD-STD-2167A DID data sources for metric questions to be both difficult and frustrating. Clearly, a contributing factor was the quality of many of the questions. The other major factor which contributed to the task's difficulty was the structure of the DOD-STD-2167A DIDs and the lack of interpretative and guidance material.

We also detected several basic incompatibilities between the Worksheets and DOD-STD-2167A. For example, Worksheet 0 contains many questions related to system-wide technical requirements, but the structure and specifications of the System/Segment Specification document did not provide obvious locations for the recording/reporting of such information.

We also developed four general recommendations as a result of our experiences in creating the cross reference guide:

1. Develop a metric questions guidebook.

2. Develop a DOD-STD-2167 guidebook for users of the Methodology.

3. Require future development of metric questions to follow a standard process.

4. Submit modification requests for DOD-STD-2167A.

5. SUMMARY

This effort involved a comprehensive review of the RADC Software Quality Measurement Methodology and the related government software development and quality standards. Our objective in conducting this investigation was to provide enhancement recommendations in several areas. During the course of our work, we repeatedly concluded that aspects of the software quality measurement methodology were overly complex. As a result of this conclusion, we have included here a preliminary set of thoughts related to a new view of the RADC measurement methodology.

6. REFERENCES

[1] T.P. Bowen, G. B. Wigle, and J. T. Tsai "Specification of Software Quality Attributes," RADC-TR-85-37, Rome Air Development Center, Feb., 1985

[2] James L. Warthman, "Software Quality Measurement Demonstration Project I," Final Technical Report, RADC-TR-87-247, Dec., 1987

[3] Patricia Pierce, Richard Hartley, and Suellen Worrells, "Software Quality Measurement Demonstration Project II," Final Technical Report, RADC-TR-87-164, Oct., 1987

[4] Dynamics Research Corporation, "Guidebook Validation Results," Sept., 1986, Contract no, F19628-84-D-0016

[5] AFSC Pamphlet 800-43, Software Management Indicators, Jan. 1986

[6] AFSC Pamphelt 800-14, Software Quality Indicators, Jan., 1987

[7] DOD-STD-2167A. *Military Standard Defense System Software Development*, Feb.,1988

[8] DOD-STD-2168. *Military Standard Defense Software Quality Program*, April, 1988

[9] Scientific Applications International Corp and Software Productivity Solutions, Inc., "Software Quality Framework Issues," TR (Interim) Volumes I, II, III, IV, 2 June 89, prepared for Rome Air Development Center, Contract no. F30602-88-C-0019.

[10] SDS Documentation Set, June, 1985, Rome Air Development Center

[11] Michael S. Deutsch and Ronald R. Willis, *Software Quality Engineering*, Prentice Hall, 1988

SECTION II

ELEMENTS of a SOFTWARE QUALITY COMPLIANCE MODEL

1. INTRODUCTION

Starting in 1976, RADC has been funding the development of a software quality measurement methodology. During this time and up to the present, this research program has been the only sustained effort which has attempted to define a methodology to specify and evaluate software quality at the product level. The current definition of the methodology is found in [1]. The key underlying concept of the methodology is a 3 level hierarchial model. The top level is a set of customer-oriented *software quality factors*. The second level is a larger set of defining attributes for the software quality factors. These are termed *software quality criteria* and reflect technical considerations of good software engineering and development practice. The third level is a still larger set of *software quality metrics* which are believed to be valid component measures of the software quality criteria. The software quality metrics in turn are defined by lower level metric elements.

Although the conceptual basis of the model is straightforward, the application of the model and the methodology to actual software development efforts is non-trivial, both in terms of conceptual application and procedure [2,3,4]. The model and the methodology have generated much discussion and debate during the past several years. Most recently, a group of RADC contractors formed a Software Quality Methodology Working Group to comprehensively review the model and the methodology[1]. As our own work on this effort proceeded, including attendance at the Working Group Meetings, we concluded that the methodology was unnecessarily complex relative to the goal of improving the quality of delivered software. Our concern was that validation of the model and the methodology could be intractable. If this turned out to be the case, widespread acceptance of the methodology by the contracting community would be impaired or possibly precluded. We also strongly believed and continue to feel that a practical software quality measurement methodology has enormous potential to improve the quality of software developed by the contractor community. Thus, we set out to interject a different perspective, based on prior and current RADC supported efforts, in the hopes of stimulating a complementary direction for future work. It is not our purpose here to analyze in any detail the current model and methodology. The following sketches out our preliminary thoughts, admittedly in too little detail.

2. AGGREGATE SCORING VERSUS REQUIREMENTS COMPLIANCE

Guidebook Volume II provides a methodology for specifying software quality goals. The end result of the specification process is a set of quantitative goals for each of the top level software quality factors. Guidebook Volume III provides a methodology for evaluating achieved levels of software quality for intermediate and final software products. The evaluation is based on answers to the low level metric element questions. Most of these questions generate values of 1 if Yes and

---

[1] Organizations represented in the Working Group included Computer Science Innovations, Inc. (CSI), the Data and Analysis Center for Software (DACS), Dynamics Research Corporation (DRC), Illinois Institute of Technology Research Institute (IITRI), RADC, RIT, Scientific Applications International Corporation (SAIC), Software Productivity Solutions, Inc. (SPS), and US Army Communications-Electronics Command (CECOM). The results of the Working Group's meetings are found in [5].

0 if No. These values are then aggregated to generate metric scores which are in turn aggregated to generate criteria scores. The criteria scores are then aggregated to generate the factor scores which are compared to the factor goals. We have termed this approach the *Software Quality Aggregate Scoring Model.*

From the academic point of view, we are concerned that the compound effects of score averaging, weighting and aggregation will hinder efforts to validate this model. Also, we believe parts of the methodology are unnecessarily complex relative to the goal of improving the quality of delivered software. Our research contractor colleagues agree with this assessment. Unfortunately, demonstrated validity will be a critical success factor in the eventual acceptance of the methodology by the contracting community. Our concerns over this matter have lead us to propose a more simple and focused model we have named the *Software Quality Compliance Model.*

The key idea for the compliance model was suggested by work done at Hughes Aircraft Company by Deutsch and Willis [6]. They asserted that the Guidebook Volume III metric questions should be rephrased in terms of testable requirements. With this perspective in mind, we conceptualized a two-level model by eliminating the middle level criteria level from the RADC model. The top level quantitative factor goals are replaced by a set of required (mandatory) software features. The extent of the features required for a given quality factor is dependent on the desired quality level for that factor. Assessment of achieved quality is a matter of determining the percentage of required software features actually implemented. Hence, the term compliance model. We also see a real need to provide a technical assessment of implementation (source code) quality. We envision this being accomplished automatically by a quality-oriented source code analyzer.

3. A SOFTWARE QUALITY COMPLIANCE MODEL

3.1. Quality Factor Specification

In the compliance model, quality factors are not assigned quantitative goals. In their place, a set of software feature requirements are attached to each quality factor selected for inclusion in the specification phase of the methodology. The extent of the requirements set, in terms of the number of requirements and the level of specificity of each requirement, is a function of the required quality level for that factor. Quality levels are expressed on a simple scale such as Ultra High, Very High, High, Average and Moderate. An immediate problem is: how is the requirement set generated for each quality factor?

Current RADC supported work provides a powerful solution to this problem. One part of the solution is to rely on domain analyses which are conducted specifically to correlate system or functional requirements with software quality requirements. Preliminary work in several mission areas is underway [7]. The other part of the solution is an expert system, designed to support the software quality measurement methodology, which incorporates the domain analysis into the system's knowledge base. The presence of domain knowledge, together with more general software engineering knowledge, will permit the expert system to not only support the generation of the quality factor requirement sets but also support the necessary technical tradeoff analysis and eventually the cost tradeoff analysis as well. In addition, the expert system can support the identification of the most critical quality factors for the system under consideration. Such an expert system has been under development for several years [8].

3.2. Evaluation of Achieved Quality

The evaluation of achieved quality has two components. The first determines the extent to which the contractor has complied with the requirements set for each factor. That is, what percentage of

the requirements set actually has been implemented in the software being reviewed or being delivered. The result of this determination is simply expressed as a compliance percentage and is reported along with a list of the requirements not implemented. We leave as open issues two items. First, what methodology is used to precisely determine which requirements have been implemented. Second, what uses can be made of the compliance report.[2]

The second evaluation component is a determination of the technical quality of the implemented software requirements. We equate technical quality with adherence to software engineering principles and best current practice. Since this evaluation component is focused on the source code, the analysis can be performed automatically by an appropriate source code analyzer.[3] The results of the source code analysis can, of course, be reported at several different levels of detail.[4] In keeping with the spirit of this discussion, we envision a single figure-of-merit (together with a simple relative scale), being reported to give the acquisition manager and/or customer a sense of the quality of the technical work.

4. ASSESSMENT

Although we have here only sketched out our preliminary thoughts, some assessment of the compliance model seems appropriate. Following we give some indications of the model's advantages and disadvantages.

4.1. Advantages

1. Conceptual Simplicity. It appears that the compliance model is easy to understand and we think would make sense to a software acquisition manager, a development contractor, the customer and the user(s).

2. Focus. We believe that focus on requirements is a major advantage of the compliance model. It is immediately of interest to the software acquisition manager, to the customer and to the user(s).

3. Reporting Structure. The minimal reporting structure is the compliance report and the technical implementation figure-of-merit. We believe that even this minimal reporting structure has potentially significant management value, while at the same time it is both streamlined and easy to understand. It largely avoids questions of statistical validity and statistical interpretation.

4. Contractor behavior. The compliance model does not appear to have elements which could be advantageously manipulated to contractor benefit.

---

[2] Most likely, some method of lifecycle requirements traceability would be used, as is the case in the current RADC methodology. As for the compliance report, the contractor would probably be required to explain and justify why certain requirements were not implemented. A mechanism would then be required to assess the severity of non-compliance.

[3] One such source code analyzer for the Ada programming language is the ADAMAT ™ product developed by Dynamics Research Corporation, Andover, MA..

[4] Very detailed analysis could be provided to programmers while a more aggregate level of detail would be useful to programming managers.

5. Training effort. The simplicity of the compliance model, together with the absence of an elaborate evaluation system based on scoring, should mitigate the amount of training required

4.2. Disadvantages

1. Dependence on domain analysis. The effectiveness of the compliance model rests largely on the accuracy and completeness of the requirements set. This, in turn, is dependent on the availability of a software quality-oriented domain analysis. Such domain analyses are rare and expensive to produce.

2. Lack of developmental support. The suggested reporting structure does not provide information which can be used to support identification and correction of software development problem areas. Modification of the reporting structure to meet this concern may impair some of the model's advantages.

5. CONCLUSION

We anticipate that a fully developed Software Quality Compliance Model would gain wide acceptance in the software development contractor community, while at the same time provide the government with an effective software quality management tool. While the compliance model can be viewed as an alternative to the current RADC aggregate scoring model, we believe the better perspective is the one that views the two models as complementary. In fact, both models could be used simultaneously. The compliance model would be used for high level management control and review, while the aggregate scoring model would be used to identify and support development problem resolution.

6. REFERENCES

[1] T.P. Bowen, G. B. Wigle, and J. T. Tsai "Specification of Software Quality Attributes," RADC-TR-85-37, Rome Air Development Center, Feb., 1985

[2] James L. Warthman, "Software Quality Measurement Demonstration Project I," Final Technical Report, RADC-TR-87-247, Dec., 1987

[3] Patricia Pierce, Richard Hartley, and Suellen Worrells, "Software Quality Measurement Demonstration Project II," Final Technical Report, RADC-TR-87-164, Oct., 1987

[4] Dynamics Research Corporation, "Guidebook Validation Results," Sept., 1986, Contract no, F19628-84-D-0016

[5] Scientific Applications International Corp and Software Productivity Solutions, Inc., "Software Quality Framework Issues," TR (Interim) Volumes I,II,III,IV, 2 June 89, prepared for Rome Air Development Center, Contract no. F30602-88-C-0019.

[6] Michael S. Deutsch and Ronald R. Willis, *Software Quality Engineering*, Prentice Hall, 1988

[7] Advanced Technology, Inc., "Mission Area Analysis," Contract no. F30602-87-D-0094.

[8] Dynamics Research Corporation, "Operational Concepts Document for the Assistant for Specifying the Quality of Software (ASQS)," 23 Oct 1987

SECTION III

STUDY of the SOFTWARE QUALITY SPECIFICATION PROCESS

1. INTRODUCTION

The Rome Air Development Center has sponsored ongoing research into a software quality specification and measurement methodology, hereafter referred to as the "Methodology" The Methodology is documented in a three volume set of guidebooks:

— *Specification of Software Quality Attributes*, RADC-TR-85-37, Vols. I–III. [1]

Volume I summarizes the DOD system acquisition process and discusses in some detail the interrelationships between software acquisition and system acquisition. Volume II presents a methodology for identifying system functions to be implemented in software, determining the relative importance of a set of software quality factors, the technical relationships (tradeoffs) between these factors, and the cost relationships (tradeoffs) between these factors. The end result of applying the specification methodology is a set of quantitative software quality goals which reflect the impact of technical and cost constraints on desired levels of software quality factors. Volume III presents a software quality evaluation methodology for determining the achieved levels of the software quality factors in intermediate and final software products.

This paper presents our analysis of the software quality specification process outlined in Guidebook Volume II and describes our recommendations for changing that process.

A summary of our analysis is as follows. The current software quality specification process is too detailed and is not validated. It is too detailed because it requires specifying separate software quality factor goals for each identified software function, whereas we believe it is more appropriate to specify overall system quality factor goals or, at most, CSCI-level quality factor goals. It is also too detailed in that the system acquisition manager must devote significant amounts of time to determine what are meant to be broad quality goals; the amount of work required is out of proportion with the scope and accuracy of the results. The current process is also unvalidated. In particular, the negative factor interrelationships and incremental quality factor costs are based only on opinions and experience, not on actual real-world project data.

A summary of our recommendations is as follows. The software quality specification process is simplified by specifying quality factor goals at the system level, or at most the CSCI level. The process is also simplified by eliminating the tailoring of metric and criterion weights; all metric questions for a criterion and all criteria for a factor are weighted equally. The feasibility of meeting all the quality factor goals is studied as part of a risk analysis, which we recommend be conducted in accordance with AFSC Pamphlet 800-45, *Software Risk Management*. [2] Software quality risk has two components: technical risk and cost risk. We suggest that the technical risk assessment be based on a simplified analysis of negative factor interrelationships. We also suggest that the cost risk assessment be based on an analysis of incremental costs derived from the COCOMO software cost model, which is a validated model based on real-world data [9] The software quality risk becomes one part of the overall risk analysis and may be handled in various ways, including assuming the risk and making no changes in the quality factor goals, or avoiding the risk by changing some of the quality factor goals.

Section 2 critiques the software quality specification process as described in Volume II. Section 3 gives an overview of our recommended software quality specification process, while Section 4 describes our recommendations in detail.

2. THE CURRENT SOFTWARE QUALITY SPECIFICATION PROCESS

The single biggest criticism we and others have of the current specification process is that it is too complicated. Our overall goal in this study has been to simplify the specification process while retaining the valuable software quality information it provides. In this section we critique the current specification process and indicate areas where it can be simplified and improved. In Sections 3 and 4 we describe the revised specification process.

2.1. Automated Tools and the Methodology

It might be argued that automating the Methodology through the use of software tools would overcome the Methodology's problem of being too complex. Indeed, two tools have been proposed and are being developed: the Assistant for the Specification of Quality Software (ASQS) [3], an expert system that helps the acquisition manager carry out the Volume II quality specification process, and the Quality Evaluation System (QUES) [4], a tool that supports the Volume III evaluation process (collection of quality evaluation data from project documents and source code, and calculation of metric, criterion, and factor scores). Future versions of the ASQS will assess compliance with specification goals using the data collected and processed by the QUES.

While we agree that these automated tools are useful and valuable, we still recommend that the Methodology be simplified. Because of potential defects or oversights in the tool, the results it produces may look reasonable, yet be incorrect. The tool user—the acquisition manager in this case—must in principle be able to check the tool's results by following the Methodology manually. If the Methodology is too complicated to be carried out manually, the acquisition managers will be forced to rely on the tools' results without being easily able to check them independently.

Therefore, the discussion below is written with the viewpoint that the Methodology must, at least in principle, be able to be carried out manually. Points at which the ASQS and QUES tools will assist the acquisition manager are noted, but these tools are not emphasized.

2.2. Software Quality Factors

The Methodology defines 13 software quality factors:

— Performance factors:

  Efficiency
  Integrity
  Reliability
  Survivability
  Usability

— Design factors:

  Correctness
  Maintainability
  Verifiability

— Adaptation factors:

  Expandability
  Flexibility
  Interoperability
  Portability
  Reusability

Substantial disagreements exist related to the appropriateness of retaining some of these factors in future revisions of the Methodology, as well as the precise definitions of the factors and of their underlying technical attributes. Comprehensive treatments of these and related issues are contained in [5,6,7].

2.3. Setting Quality Factor Goals

Volume II requires the acquisition manager to identify the major software functions in the system being acquired and to set quality goals separately for each function. Alternatively, the acquisition manager could set quality goals for the entire system rather than for each function. Below we discuss the pros and cons of these two approaches. We recommend changing the Methodology to use the latter approach.

Specifying quality goals separately for each function has the advantage that different functions in the software can get different sets of quality goals, depending on each function's criticality. This in turn allows the developers to target their efforts; they can spend more resources on the functions that require higher quality levels than on those that require lower quality levels. Generic domain analyses would list the functions and their recommended quality levels for different kinds of systems.

Specifying quality goals separately for each function has many drawbacks, though:

- It may be unclear what the "functions" are in a particular system, especially if no generic domain analysis is available. The acquisition manager must then define the functions arbitrarily, and there is no guarantee that the metric values achieved by this system's functions will be comparable to any other system's functions. Specifying quality goals and collecting metric data for the system as a whole ensures that the data can be compared with, and the experience with this system can be applied to, other similar systems.
- The system's users usually do not perceive variations in quality requirements among the system's functions—if indeed they are aware of separate functions at all. They say, "This weapon must be highly reliable, highly maintainable," and so on. Specifying quality goals for the system as a whole is more in accord with the users' viewpoint.
- Under DOD-STD-2167A [10], all software documentation is organized by the Computer Software Configuration Item (CSCI), not by the function. A CSCI tends to be a large conglomeration of many functions. If the documentation is organized by CSCI rather than by function, it makes more sense to organize the quality goals the same way—to specify quality goals for the system as a whole, or at most for each CSCI.
- Each CSCI is further organized into Computer Software Components (CSCs). It is therefore easy to collect metric data separately for CSCs. However, there is no clear relationship between CSCs and functions. It is likely that each function will be implemented by several CSCs, and each CSC will implement part of several functions. It is unclear how to allocate a CSC's metric scores among the functions it helps to implement. Specifying quality goals for the system as a whole avoids these problems: all the CSCs' metric scores are combined into a single system-wide score.
- If the Methodology is to be accepted it must be usable. Requiring acquisition managers to deal with each function separately—specifying quality goals, collecting metric data, and calculating metric scores separately for each function—makes implementing the Methodology too much work. Implementing the Methodology at just the system level is a substantial undertaking.

Specifying quality goals for the system as a whole, or at most for each CSCI, rather than for each function, will greatly simplify the Methodology and make it more usable. We recommend changing the Methodology in this way.

2.4. Complementary Factor Interrelationships

Volume II Section 4.1.2.4 requires the acquisition manager to consider complementary factor interrelationships when setting quality factor goals. The factors that are complementary to most other factors are Reliability, Correctness, Maintainability, and Verifiability. If a high level of quality is specified for certain factors, say Reusability, then a high level of quality must also be specified for the four complementary factors. The reasoning is that a project that scores high on the Reusability factor but low on a complementary factor like Maintainability cannot truly be highly reusable.

In order to simplify the Methodology, we agree with others [5,7] that consideration of the complementary factor interrelationships should be eliminated. If there are indeed complementary relationships among quality factors on certain kinds of systems, then the generic domain analysis for that kind of system should recommend that high quality levels be specified for all the related factors. This accounts for the interrelationships implicitly during the initial quality goal setting rather than explicitly as a separate step in the process.

2.5. Positive and Negative Factor Interrelationships

Volume II Section 4.1.3 discusses positive and negative interrelationships among the quality factors. Negative factor interrelationships are problematic; if high quality goals are specified for two negatively interrelated factors, achieving high quality for both of them will be more difficult. Section 4.1.3 offers several options for dealing with conflicts: try to achieve the quality goals anyway, reduce the goal for one of the factors, or increase the computing hardware capabilities. Although Section 4.1.3 does not speak in these terms, this is very much like a risk analysis. We recommend that consideration of negative factor interrelationships take an explicitly risk-based approach. The more pairs of factor goals that are negatively interrelated and that are both specified as high, the greater the technical risk that those goals cannot be achieved.

This software quality technical risk is only one of many risk areas the acquisition manager must assess. DOD-STD-2167A requires that a risk assessment be done early in the project, and methods for managing the risk be documented and followed. AFSC Pamphlet 800-45 gives a procedure for assessing risk and deciding how to manage it. We recommend that consideration of negative factor interrelationships be one more item in the total risk assessment.

With a risk-based approach, the acquisition manager has several options described in AFSC Pamphlet 800-45 if a high-risk set of software quality goals is specified. The acquisition manager can change the software quality goals to reduce the risk (risk avoidance). However, the acquisition manager can also decide that, in the context of the entire risk assessment, the software quality risk is acceptable, and leave the software quality goals unchanged (risk assumption). The acquisition manager can implement some requirements in hardware rather than software, thus reducing the software risk (risk transfer).

2.6. Cost Analysis

After setting quality factor goals, the acquisition manager analyzes how much meeting them will cost. Volume II Section 4.1.4 gives a procedure for estimating the incremental software quality cost: the extra percentage of the base software development cost that must be expended to achieve the software quality goals. Each quality factor goal level translates to a percentage, and the percentages are added to get an overall incremental cost.

The approach of estimating an incremental software quality cost is sound. However, the particular numbers and method in Section 4.1.4 are unvalidated. There is no real world project data from which these numbers were derived. There is no reason to believe that this method gives accurate estimates. We recommend that it be rejected. In its place, we recommend that the Methodology use an incremental software quality cost estimation model based on the Intermediate COCOMO Model.

As with the positive and negative factor interrelationships, we further recommend that the incremental software quality cost estimate become part of the overall project risk assessment. Higher costs translate to higher risks. The acquisition manager has the option of changing the quality goals to reduce the cost risk, or leaving the quality goals as they are and assuming the higher risk.

2.7. Criteria and Metric Question Weighting

Volume II Section 4.2.2 has the acquisition manager define formulas for the criteria scores as a weighted sum of the relevant metric question scores, and formulas for the factor scores as a weighted sum of the relevant criteria scores. The acquisition manager is allowed to assign unequal weights to different components of a formula—perhaps because certain metric questions or criteria are less important than others in a particular application.

However, as stated in [7], allowing different projects to use different sets of weights for their scores prevents meaningful comparisons among the projects' metric scores and makes it impossible to accumulate information from many projects about the levels of metric scores to expect. Also, no guidance is given on which weights to change and how much to change them.

With the goal of simplifying the Methodology, we recommend (as does [7]) that all metric questions be weighted equally in a criterion's formula, and all criteria be weighted equally in a factor's formula. At some point in the future, the knowledge base of the ASQS may be sufficiently mature as to permit objective weighting suggestions

2.8. Numerical Factor Goals

Volume II Section 4.1.4.4 recommends that the final quality factor goals be translated to ranges of factor scores this way: E—from 0.90 to 1, G—from 0.80 to 0.89, A—from 0.70 to 0.79. If a project's final score calculated for each factor falls within the range corresponding to its goal, the goal is met, and the acquisition manager will accept delivery; otherwise the project has an unacceptable level of quality.

Like the Methodologies current cost analysis procedure (see Section 2.5), these ranges are unvalidated. Validating these ranges would involve taking a large number of real-world software projects which have been demonstrated to have high quality (say, acceptably low defect rates during the first year of operation), computing the quality factor scores for those projects, and determining if the factor scores fall within the correct numerical ranges. No such study has been done.

However, the acquisition manager must be able to decide if the quality factor scores calculated from the final software meet the quality factor goals specified for the project. The only way to do this is by assigning a factor score range to each quality goal level. In the absence of any real-world data, we recommend that the following ranges be used for now: E—0.90 to 1.00, G—0.80 to 1.00, A—0.70 to 1.00. Note that we have expanded the upper end of the G and the A ranges to 1.00. This is because a calculated factor score of, say, 0.95 does meet a factor goal of, say, G. (If a factor score is sufficient to meet a certain goal, it must *a fortiori* be sufficient to meet any lower goal.)

As the Methodology is applied to actual projects and data accumulates, these ranges must be validated and adjusted if necessary.

3. OVERVIEW OF THE RECOMMENDED PROCESS

Figure 1 is a flowchart of the software quality specification process we recommend to replace the one in Volume II. This section summarizes the steps in the process. Details of each step are given in Section 4. Our work has been influenced by Deutsch's and Willis's approach in [8].

Eliminate Factors. The acquiring manager begins by tailoring the Framework. This consists of eliminating entire quality factors that are not applicable to the system being acquired.

Set quality factor goals. For each remaining quality factor, the manager sets a goal of E (excellent), G (good), or A (average) quality. This is done by a user survey and/or by referring to a generic problem domain analysis.

Analyze technical risk. The manager analyzes possible negative interrelationships between the goals set for the quality factors and determines the risk that the specified combination of goals will not be met.

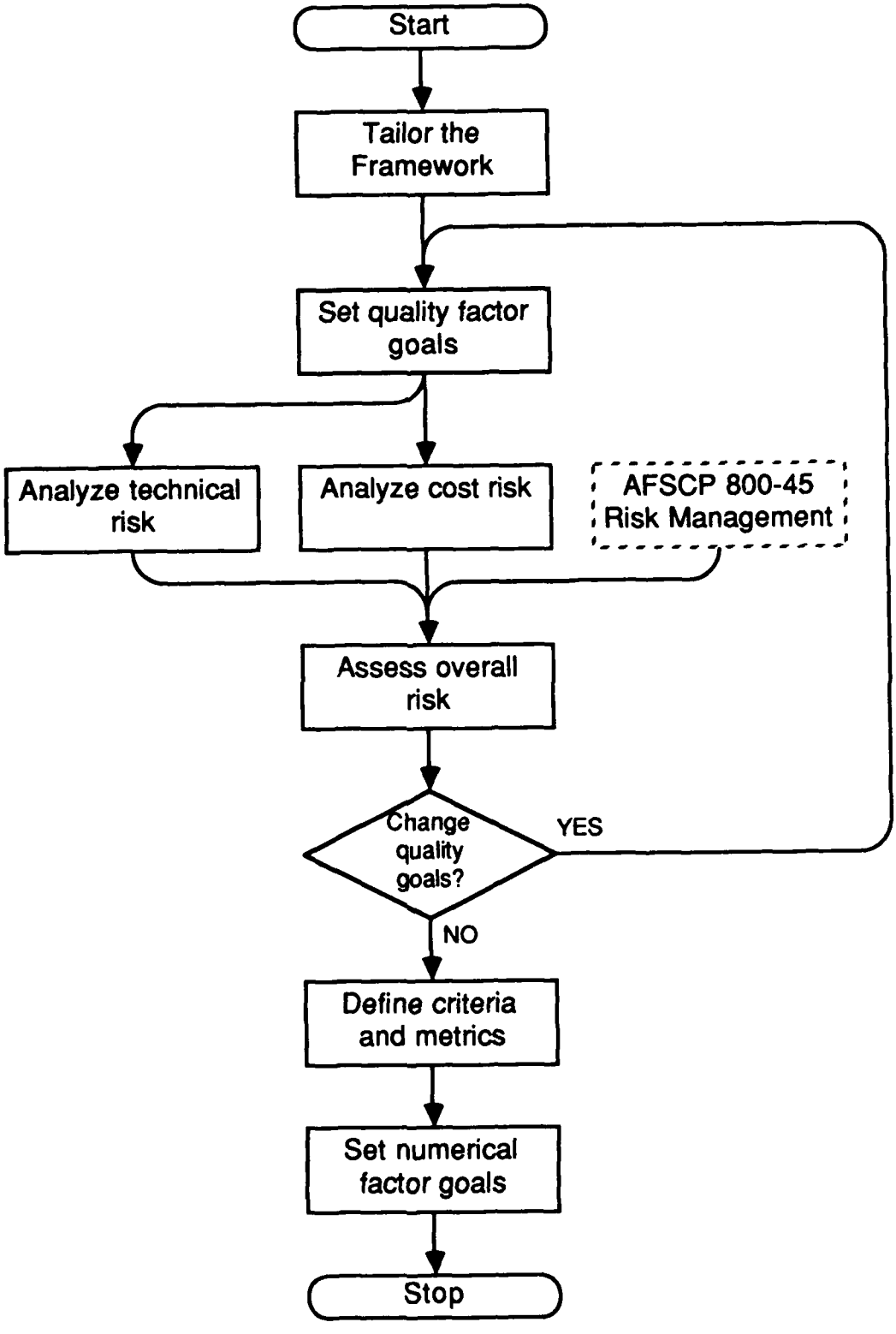
Analyze cost risk. The manager estimates the cost to achieve the goals set for the quality factors relative to the development cost without those goals. This takes the form of an incremental cost; for example, achieving the quality goals will increase the development cost by 10%.

Assess overall risk. The technical and cost risk analysis results are folded into the overall software risk assessment described in AFSC Pamphlet 800-45. This pamphlet recommends that the program office implement risk management studies very early in the acquisition cycle, during mission analysis and concept exploration. This is the time when the acquiring manager would be establishing software quality requirements using the Framework.

The assessment includes deciding how to handle the risk that exists, if any. One way to handle the risk is simply to "assume" the risk (gamble that the system will be developed successfully); then the quality goals do not need to change. Another way to handle the risk is to "avoid" it by changing the quality goals to reduce the risk; this requires re-analyzing the risk.

Define criteria and metric questions and weights. At this point the manager has a set of quality factor goals whose risk is acceptable. All that remains is to prepare quality data collection worksheets that contain all metric questions for all criteria that are part of the specified factors, and to define formulas for calculating criteria scores from the metrics and factor scores from the criteria. To facilitate comparing factor scores between projects, all metrics and criteria are weighted equally.

Figure 1. Recommended Software Quality Acquisition Process.



Set numerical factor goals. Finally, the manager translates the quality factor goals (E, G, and A) into numerical factor score ranges. Before the program office will accept the system, the software's computed factor scores much all fall in their required ranges, or the ranges must be changed.

The ASQS tool will assist the acquisition manager to carry out this software quality acquisition process.

4. DETAILS OF THE RECOMMENDED PROCESS

4.1. Eliminate Factors

To begin the software quality acquisition process, the acquiring manager eliminates those quality factors that are not applicable to the system being acquired.

The manager can draw on several sources of guidance when deciding which factors to retain and which to eliminate:

— *Acquisition documents*. Requests For Proposal, Statements of Work, and other such documents may already state software quality requirements, directly or indirectly, that can be used to determine which quality factors are needed.

  Table 4.1.2-3 in Volume II is useful when studying acquisition documents. This table shows the correlation between 16 general system quality factors and the Methodology's quality factors.

— *Generic problem domain analyses*. A generic description of a weapon system, a C^3I system, or any other kind of system would supply not only a list of functions all such systems perform, but also an indication of areas that do and do not require high software quality.

  For example, some hypothetical weapon system software installed in a missile might require high reliability (so as not to fail in flight), but might not be concerned with integrity (because no one will be trying to break into the system). A C^3I system might require high integrity (to prevent malicious users from breaking into the system), but might not be concerned with portability (because the user will never require it to run on a different hardware configuration).

  Table 4.1.2-1 and Table 4.1.2-2 in Volume II are a rudimentary form of such a problem domain analysis. The former table lists important software quality factors for major command and control applications. The latter table lists examples of application and environment characteristics and their related software quality factors.

— *User surveys*. The user surveys described in Section 4.2 can be used not only to determine the users' desired quality goals, but also to find out which quality factors the users feel are not applicable. Such factors can be tailored out.

— *Maintainer surveys*. Similarly, the personnel who will be maintaining the system after delivery should be surveyed.

— *The manager's own experience* in acquiring similar systems.

We believe that certain quality factors—Correctness, Maintainability, and Verifiability—are mandatory and must never be eliminated. This is because a lack of concern for these attributes is poor software engineering practice, no matter what kind of system is being developed.

4.2. Set Quality Factor Goals

Having determined which quality factors are applicable, the manager next sets a qualitative goal for each quality factor. The possible goals are:

— E: Excellent quality is required for this factor.

— G: Good quality is required for this factor.

— A: Average quality is required for this factor.

— N/I: Quality is not an issue for this factor.

If the software consists of a single CSCI, as is often the case, then the quality goals are set for the software as a whole. If the software consists of more than one CSCI, then the quality goals can be set either for the software as a whole, or for each CSCI individually. Quality goals are not set for each function, as is currently prescribed in Guidebook Volumes I and II.

When setting quality factor goals, the manager can draw on the same sources of guidance as described in Section 4.1:

— *Acquisition documents*. Requests For Proposal, Statements of Work, and other such documents may indicate the level of quality needed.

— *Generic problem domain analyses*. A generic description of a weapon system, a C^3I system, or any other kind of system would indicate the quality level, or range of quality levels, required for each quality area.

— *User surveys*. The manager should survey the system's users to determine their quality needs. The quality requirements survey should take the form described in Volume II, Paragraph 4.1.2.3, Table 4.1.2-4, and Table 4.1.2-5, except that the user would fill out quality requirements for the entire system or for the CSCIs rather than for each function.

— *Maintainer surveys*. Similarly, the personnel who will be maintaining the system after delivery should be surveyed.

— *The manager's own experience* in acquiring similar systems.

As discussed previously in Section 2.3, complementary factor interrelationships have been taken out and do not influence setting the quality factor goals.

4.3. Analyze Technical Risk

Having set the quality factor goals, the manager proceeds to analyze the risk associated with that set of goals; that is, the probability estimate that the desired set of quality goals will not be met. The software quality risk is analyzed in two areas, software quality technical risk (described in this section) and software quality cost risk (described in Section 4.4). The results of these risk analyses are factored into an overall software risk analysis (described in Section 4.5), and the manager then decides how to handle the risk.

4.3.1. Factor Interrelationships

The technical risk analysis is based on the factor interrelationship matrix in Figure 2. This is taken from Volume II, Table 4.1.3-4, with the Efficiency and Interoperability factors removed. (Recent critical analyses of the Methodology [5,7] have recommended that these two factors be eliminated). Compared to the original Table, it will be seen that a significant reduction in the number of factor interrelationships is accomplished. We believe this is a superior state of affairs, since the overall complexity of analyzing technical risk has been reduced.

Figure 2. Quality Factor Interrelationships.

	Integrity	Reliability	Survivability	Usability	Correctness	Maintainability	Verifiability	Expandability	Flexibility	Portability	Reusability
Integrity											
Reliability				+							
Survivability	–			+		+		–	–	–	–
Usability						+	+				
Correctness						+	+	+	+		+
Maintainability							+	+	+		+
Verifiability											
Expandability	–	–	–								
Flexibility	–	–	–								
Portability											
Reusability	–	–	–			+				+	

The entries in the matrix are interpreted as follows:

Entry	Interrelationship
+	Positive interrelationship. Raising the level of quality for the factor in the row generally results in raising the level of quality for the factor in the column.
–	Negative interrelationship. Raising the level of quality for the factor in the row generally results in reducing the level of quality for the factor in the column.
Blank	No interrelationship. Raising the level of quality for the factor in the row generally has no effect on the level of quality for the factor in the column.

For example, the matrix asserts that software which has a high degree of correctness is also likely to have high maintainability as well (software without bugs is easier to maintain). Software with high survivability is likely to have low portability (techniques for recovering from system failures tend to be highly machine-specific and non-portable).

A technical risk may be present if these two conditions are both true:

— Two quality factors have a negative interrelationship.

— The quality goal for each of these factors is consequential (i.e., E, G, or A.

In such a case, the interrelationship matrix asserts that there is a good chance that the two consequential quality goals cannot be met simultaneously; the risk is high. On the other hand, note

that if the one goal is E, G, or A, and the other goal is N/I for these negatively-interrelated factors, then those goals are in accordance with the matrix and there is little risk. For positively-related or non-related factors, if both goals are consequential, then they are also in accordance with the matrix and there is little risk.

4.3.2. Software Quality Technical Risk Analysis Procedure

From these considerations, the technical risk analysis procedure is as follows.

1. Consider a pair of quality factors that have a negative interrelationship, as shown in Figure 2.
2. Decide if a risk exists based on the quality goals for these factors.
   a. If the goal for the factor in the row in Figure 2 is E, and the goal for the factor in the column in Figure 2 is E, G, or A, then a risk exists for this pair.
   b. If the goal for the factor in the row is G, and the goal for the factor in the column is E or G, then a risk exists for this pair.
   c. If the goal for the factor in the row is A, and the goal for the factor in the column is E, then a risk exists for this pair.
   d. Otherwise, a risk does not exist for this pair.
3. Repeat Step 2 for all remaining pairs of quality factors that have a negative interrelationship.
4. Assign an overall software quality technical risk level.
   a. If there are no risky pairs of quality factor goals, then the overall software quality technical risk level is *low*.
   b. If there are 1 to 3 risky pairs of quality factor goals, then the overall software quality technical risk level is *medium*.
   c. If there are 4 or more risky pairs of quality factor goals, then the overall software quality technical risk level is *high*.

The final software quality technical risk level, low, medium, or high, becomes part of the overall software risk analysis described in Section 4.5.

The above ranges of numbers of risky pairs are suggestions. The acquiring manager must exercise his or her own judgment in deriving a risk level from the number of risky pairs. Also, generic domain analyses would identify specific risky pairs that are important for different mission areas.

4.4. Analyze Cost Risk

After the software quality technical risk, the manager analyzes the software quality cost risk. The manager first estimates the cost to achieve the goals set for the quality factors relative to the development cost without those goals. The result is an incremental cost; for example, achieving the quality goals will increase the development cost by 10%. This incremental cost is then used to assign a risk level.

Our incremental quality cost estimation approach is based on Deutsch's and Willis's approach, which in turn relies on the COCOMO software cost estimation model. Our justifications for basing quality cost calculations on the COCOMO model are:

— COCOMO is the most widely used cost estimating model in the private and government software development communities.

— This wide body of experience has shown that the COCOMO model yields acceptably accurate estimates on the order of +/- 20% approximately 70% of the time.

— The COCOMO model is undergoing continual improvement and revision as experience with it accumulates. Barry Boehm at TRW has recently revised the cost driver component of the model and has provided a model variant specific to Ada [11].

Section 4.4.1 gives an overview of the Intermediate COCOMO Model. The remaining subsections describe how that model is used to estimate incremental software quality costs in the Framework.

4.4.1. The Intermediate COCOMO Model

First, a nominal effort estimate for the project is derived based on the project's development mode (organic, semidetached, or embedded) and the size of the software to be developed (in thousands of developed source instructions, or KDSI). For an embedded mode project, the nominal effort formula is:

$$\text{Effort (staff-months)} = 2.8\ (\text{KDSI})^{1.20}$$

Second, this nominal effort estimate is modified based on characteristics of the project. These take the form of fifteen cost drivers:

— Product attributes:

  — Required software reliability.
  — Data base size.
  — Product complexity.

— Computer attributes:

  — Execution time constraint.
  — Main storage constraint.
  — Virtual machine volatility.
  — Computer turnaround time.

— Personnel attributes:

  — Analyst capability.
  — Applications experience.
  — Programmer capability.
  — Virtual machine experience.
  — Programming language experience.

— Project attributes:

  — Use of modern programming practices.
  — Use of software tools.
  — Required development schedule.

Each cost driver is rated on a six-point scale: very low, low, nominal, high, very high, and extra high. The rating for each cost driver gives an effort multiplier for each cost driver. Here are the effort multipliers for the various ratings of selected cost drivers:

Cost Driver	Very low	Low	Nominal	High	Very high	Extra high
Required software reliability	0.75	0.88	1.00	1.15	1.40	1.40
Use of modern programming practices	1.24	1.10	1.00	0.91	0.82	0.82
Use of software tools	1.24	1.10	1.00	0.91	0.83	0.83

Third, the fifteen multipliers for all the cost drivers are multiplied together to give an overall effort multiplier; this can range anywhere from 0.09 to 72.97. Then the overall effort multiplier and the nominal effort estimate are multiplied to give the actual effort estimate.

Finally, a schedule estimate and a breakdown of effort and schedule within each development phase are computed from the actual effort estimate. Since we are interested in estimating just the cost of software quality, and since the cost of a software project is determined by the total effort in staff-months rather than the schedule, the schedule and phase breakdown formulas will not be discussed here.

4.4.2. Rationale for the Software Quality Cost Analysis Approach

The quality requirements influence the development cost in three ways:

— Increased software size.

— Increased cost drivers.

— Increased independent verification and validation (IV&V) costs.

First, specifying high quality levels may require adding code to the software that would not be needed otherwise. For example, software with high survivability must have additional fault-recovery code. This increased software size increases the KDSI input to the COCOMO model, thus increasing the cost.

The acquisition manager must therefore obtain an estimate of the percent increase in the software size that will be needed to implement the quality factor goals. This estimate can be obtained from the history of systems previously acquired, by asking software developers who have worked on similar projects in the past, or by other means.

The incremental cost due to increased software size is estimated as follows. Let E_0 be the project cost before adding the extra software to achieve the quality goals. Let E_1 be the project cost after adding the extra software. Then the incremental cost ratio is given by E_1/E_0. To calculate this, let S_0 be the software size before adding the extra software to achieve the quality goals. Let X be the percent increase in the software size required to achieve the quality goals. Then M_1 the revised cost due to increased software size, from the COCOMO formula for nominal effort, is:

$$M_1 = \frac{E_1}{E_0} = \frac{2.8\,(S_0 \times (1 + X/100))^{1.20}}{2.8\,(S_0)^{1.20}} = (1 + X/100)^{1.20}.$$

Second, some of the COCOMO cost drivers are related to software quality. Specifying high quality levels will increase the effort multipliers associated with these cost drivers, thus increasing the cost.

Only one cost driver, Reliability, directly corresponds to one of the Methodology quality factors. The specified quality goal for the Reliability factor can therefore be translated to a Reliability cost driver setting. We recommend this translation: E to very high, G to high, A to nominal, and N/I to low. From the table in Section 4.4.1, the software reliability multiplier M_2 is 1.40, 1.15, 1.00, or 0.88, respectively.

For the remaining quality factors, we assume (as do Deutsch and Willis) that the factor goals influence the Modern Programming Practices and the Use Of Software Tools cost drivers. We recommend averaging the quality goals for all factors (including Reliability), then translating this average quality goal into cost driver settings as above. If the average quality goal is E, the Modern Programming Practices and the Use of Software Tools cost drivers will both be set to very high, the numerical effort multipliers will be 0.82 and 0.83, and the combined software quality multiplier M_3 will be $0.82 \times 0.83 = 0.68$. Likewise, if the average quality goal is G, M_3 will be $0.91 \times 0.91 = 0.83$; if the average quality goal is A or N/I, M_3 will be $1.00 \times 1.00 = 1.00$.

Third, Deutsch and Willis describe a model for the additional IV&V costs that must be incurred to achieve a given level of quality. These IV&V costs would be incurred only if the software is submitted to an organization other than the developing organization for verification and validation, as part of the acquiring agency's plan for achieving the required quality goals. The additional IV&V cost is the product of two numbers: a number that depends on the average quality goal for all of the factors, and a number that is the percentage of the software subjected to IV&V. The details of Deutsch's and Willis's calculation appear in Section 4.4.3 below.

4.4.3. Software Quality Cost Risk Analysis Approach

The procedure for analyzing software quality cost risk is as follows.

1. Determine the software size multiplier, M_1.
   a. Estimate X = the percent increase in the software size (lines of code) required to implement all the quality factor goals.
   b. Calculate $M_1 = (1 + X/100)^{1.20}$.
2. Determine the software reliability multiplier, M_2.
   a. Reliability factor goal is

E.....	M_2 =	1.40
G.....		1.15
A.....		1.00
N/I.....		0.88

3. Determine the software quality multiplier, M_3.
   a. Calculate the average of the remaining ten (10) quality factor goals as follows. Assign a goal of E the number 3; G, 2; A or N/I, 1. Add these ten numbers, divide the sum by 10, and round to the nearest integer. Convert the result back to the average quality factor goal of E, G, A, or N/I.
   b. Average quality factor goal is

E	M_3 =	0.68
G		0.83
A or N/I		1.00

4. Determine the IV&V multiplier, M_4.
   a. If the software will not be subjected to IV&V, then $M_4 = 1.00$.

b. If the software will be subjected to IV&V, then M_4 depends on the average quality factor goal calculated in Step 3.a, and on Y = the percentage of the software that will be subjected to IV&V.

Average quality factor goal is		
	E..... M_4 =	$1 + (0.60 \times Y) / 100$
	G.....	$1 + (0.25 \times Y) / 100$
	A.....	$1 + (0.10 \times Y) / 100$
	N/I.....	1.00

5. Determine the estimated software cost increment in percent by calculating:

   $((M_1 \times M_2 \times M_3 \times M_4) - 1) \times 100\%$

6. Assign an overall software quality cost risk level.
   a. If the estimated software cost increment is between 0% and 20%, then the overall software quality cost risk level is *low*.
   b. If the estimated software cost increment is between 20% and 50%, then the overall software quality cost risk level is *medium*.
   a. If the estimated software cost increment is greater than 50%, then the overall software quality cost risk level is *high*.

The final software quality cost risk level, low, medium, or high, becomes part of the overall software risk analysis described in Section 4.5.

The above ranges of software cost increments are suggestions. The acquiring manager must exercise judgment in deriving a risk level from the estimated software cost increment.

The above approach assumes the software development mode is "embedded." This is true of most Air Force software. If the software development mode is not embedded, the above approach will give a upwardly biased estimate of the incremental cost.

4.4.4. Example

Consider an example of the effect of increased software size due to quality requirements. Let the software size increase by 30% to implement the quality requirements. Let the quality factor goal for the Reliability factor be E. Let the average quality goal for all the other factors be G. Let 100% of the software be subjected to IV&V. Then the individual cost multipliers are:

$M_1 = 1.37$
$M_2 = 1.40$
$M_3 = 0.83$
$M_4 = 1.25$

The total estimated software cost increment is $((1.37 \times 1.40 \times 0.83 \times 1.25) - 1) \times 100\%$, or 99%.

The ultimate result of this example cost analysis, including increased software size to implement the quality requirements, the effect of the quality requirements on the cost drivers, and the IV&V costs, is to increase the software development cost by 99%. The software is estimated to cost almost twice what it would have if the quality requirements and the IV&V had not been present.

4.5. Assess Overall Risk

If the acquiring manager is following the software risk management procedures in AFSC Pamphlet 800-45, as we recommend, when the time comes to set quality factor goals the manager will already have a grasp of the proposed system's level of risk in five risk areas: technical risk, cost risk, schedule risk, operational risk, and support risk. The software quality risk assessments described in Sections 4.3 and 4.4 simply become additional components in these risk areas.

Specifically, the software quality technical risk becomes another driver in the category of technical risk. With this addition, the technical risk drivers are:

- Requirements: complexity, size, stability, post deployment software support (PDSS), reliability and maintainability, *and additional software quality requirements.*
- Constraints: computer resources, personnel, standards, government furnished equipment and products, environment.
- Technology: language, hardware, tools, data rights, experience.
- Developmental approach: prototypes and reuse, documentation, environment, management approach, integration.

The software quality cost risk becomes another driver in the category of cost risk. With this addition, the cost risk drivers are:

- Requirements: size, resource constraints, application, technology, requirements stability, *and additional software quality requirements.*
- Personnel: availability, mix, experience, management environment.
- Reusable software: availability, modifications, language, rights, certification.
- Tools and environment: facilities, availability, rights, configuration management.

Thus, the acquiring manager considers the software quality technical and cost risks derived from the Methodology along with all the other risk drivers to assess the overall software risk, as described in AFSC Pamphlet 800-45. Once the risk is understood, it must be handled. The pamphlet lists these ways of handling technical risk ([2], paragraph 2–16):

"When understanding of the risk factors, drivers, assessments, and evaluations is obtained, a risk control course of action can then be established. Possible courses of action are:

"a. Avoidance. This might involve de-scoping the program's technical objectives to a more achievable level, changing the developmental approach, and/or the use of a pre-planned product improvement (P^3I) approach to achieving the desired technical objectives through successive enhancement to a more technically achievable baseline. [It might also involve reducing some of the software quality factor goals to eliminate negative interrelationships.]

"b. Control. The program office could apply parallel development efforts, intense management reviews, rapid prototyping, and/or conduct additional trade-off studies.

"c. Assumption. The program office can accept the technical risk involved and manage the software development effort appropriately, while remaining cognizant of the decreased probability of success.

"d. Transfer. Software requirements could be realigned through the requirements analysis process, and, in some cases, be implemented via hardware vice software. Another possibility, in some integrated systems, is to reassign the requirement to other elements of the system. Under *no*

circumstances should the software technical risk be transferred to the operational and PDSS phases of the system life cycle."

The pamphlet lists these ways of handling cost risk ([2], paragraph 2–22):

"Cost risk control should always be accomplished on a total system life cycle cost analysis basis. Possible courses of action for cost risk control include:

"a. Avoidance. Software cost avoidance is normally achieved by reducing the scope of the software development effort, obtaining additional resources, "streamlined" acquisition practices, and P^3I software development. [It might also be achieved by reducing some of the software quality factor goals to reduce the incremental software quality costs.] Extreme caution should be exercised if cost avoidance techniques such as eliminating documentation, reducing the nature and scope of test, and eliminating the software quality requirements are employed.

"b. Control. Cost control can be achieved through stringent management reviews, continuous updating of the cost estimates based upon real-time inputs of technical and schedule accomplishments, and/or conducting additional trade-off studies to control cost.

"c. Assumption. The program office can accept the cost risk involved, manage the software development process accordingly, and remain cognizant of the probability of successful program accomplishment within the available financial resources.

"d. Transfer. Software functions that have a high cost risk could be transferred to related programs that have the resources to successfully implement these functions. Under *no* circumstances should cost risk be transferred to the operational and PDSS phases of the system life cycle. Cost risk transferred to these phases is totally counterproductive to reducing life cycle costs."

Thus, as part of the overall software risk analysis, the acquiring manager will decide whether to change the software quality factor goals or to keep them as they are. In the former case the manager must go back and re-analyze the software quality technical and cost risks. In the latter case the manager can proceed.

4.6. Define Criteria and Metric Questions and Weights

At this point the acquiring manager knows which quality factors will be specified for the software and goals for each factor. Now the manager must define the criteria and metrics that will be used to calculate these factors' scores.

Table 3.2-1 of Volume II lists the criteria for each factor. The formula for calculating the factor score is simply a weighted sum of the appropriate criteria scores, with each criterion being weighted equally. For example, the Maintainability factor (MA) is composed of the Consistency (CS), Visibility (VS), Modularity (MO), Self-Descriptiveness (SD), and Simplicity (SI) criteria. Therefore, the formula for Maintainability is:

$$MA = \frac{CS + VS + MO + SD + SI}{5}.$$

The manager derives the formula for each factor that has not been eliminated (each factor whose goal is E, G, or A). These formulas then determine the minimum set of criteria needed to calculate all the factors. (If some factors are eliminated, then some criteria may not be needed either.)

Once the minimum set of criteria is known, the manager can eliminate all metric questions that do not belong to the minimum set of criteria.

4.7. Assign Numerical Factor Goals

At this point the acquiring manager has formulas for calculating all the factor scores. All that remains is to decide which numerical factor scores will count as achieving a factor goal of E, of G, and of A.

As discussed in Section 2.7, in the absence of Methodology experience on real projects, we recommend that the quality factor goals be assigned to numerical ranges as in Volume II:

- A goal of E will be achieved if the factor score lies in the range 0.90 to 1.00.
- A goal of G will be achieved if the factor score lies in the range 0.80 to 1.00.
- A goal of A will be achieved if the factor score lies in the range 0.70 to 1.00.

4.8. Results

This concludes the software quality specification process using the Methodology. The acquiring manager now has the following items that can be written into a contract:

- A set of required software quality factors and a required quality level (E, G, A) for each.
- A set of worksheets with metric questions and criteria, and formulas for calculating each quality factor score from the criteria.
- A set of numerical factor score ranges that must be achieved before the system will be accepted.

5. REFERENCES

[1] T.P. Bowen, G. B. Wigle, and J. T. Tsai "Specification of Software Quality Attributes," RADC-TR-85-37, Rome Air Development Center, Feb., 1985

[2] AFSC Pamphlet 800-45. *Software Risk Management.* June, 1987.

[3] Dynamics Research Corporation, "Operational Concepts Document for the Assistant for Specifying the Quality of Software (ASQS)," 23 Oct 1987

[4] Software Productivity Solutions, Inc., "Software Requirements Specification for the Quality Evaluation System (QUES), July, 1989.

[5] James L. Warthman, "Software Quality Measurement Demonstration Project I," Final Technical Report, RADC-TR-87-247, Dec., 1987

[6] Scientific Applications International Corp and Software Productivity Solutions, Inc., "Software Quality Framework Issues," TR (Interim) Volume II, 2 June 89, prepared for Rome Air Development Center

[7] Patricia Pierce, Richard Hartley, and Suellen Worrells, "Software Quality Measurement Demonstration Project II," Final Technical Report, RADC-TR-87-164, Oct., 1987

[8] Michael S. Deutsch and Ronald R. Willis. *Software Quality Engineering: A Total Technical and Management Approach,* Prentice-Hall, 1988.

[9] Barry W. Boehm, *Software Engineering Economics,* Prentice-Hall, 1981.

[10] DOD-STD-2167A, *Military Standard Defense System Software Development,* Feb.,1988

[11] Barry W. Boehm, "Ada COCOMO," Proceedings Third COCOMO User's Group, Software Engineering Institute, Pittsburgh, PA, Nov., 1987.

SECTION IV

STUDY of AFSC MANAGEMENT INDICATORS

1. INTRODUCTION

The Air Force Systems Command (AFSC) has published two pamphlets in the area of software quality management, hereafter collectively referred to as the "Indicators:"

— *Software Management Indicators,* AFSC Pamphlet 800-43. [1]

— *Software Quality Indicators,* AFSC Pamphlet 800-14. [2]

The Rome Air Development Center has sponsored ongoing research into a software quality specification and measurement methodology, hereafter referred to as the "Methodology" The Methodology is documented in a three volume set of guidebooks:

— *Specification of Software Quality Attributes,* RADC-TR-85-37, Vols. I–III. [3]

This study reports on the relationship between the Indicators and the Methodology and describes how and to what extent the Indicators can be incorporated into the Methodology.

A summary of our recommended approach is as follows. All 6 Software Management Indicators and 4 of the Software Quality Indicators are process-oriented and should not be directly incorporated into the product-oriented Methodology. These 10 Indicators are grouped to form the "Software Process Indicators." The remaining 3 Software Quality Indicators are product-oriented. These are grouped to form the "Software Product Indicators" and are incorporated into the Methodology. During development of a software project, the Software Process Indicators are calculated every month based on a small amount of collected process data. At the end of each development phase, a much larger amount of process and product data is collected. The process data is used to calculate the Software Process Indicators as usual, and the product data is used to answer the Methodology metric questions. The Methodology metric questions in turn are used to calculate the Software Product Indicators and the Methodology metric, criteria, and factor scores. With this combined approach, the acquisition manager obtains both an overall view of the entire project's quality (from the Indicators) and a detailed view of each individual software document's quality (from the Methodology).

Section 2 compares the philosophies behind the Indicators and the Methodology; this is needed to understand how they can be combined. Section 3 describes our approach for combining the Indicators and the Methodology and justifies it based on their characteristics. Section 4 shows in detail how to calculate the three Software Product Indicators from data in the Methodology metric questions. Section 4 also recommends changing several existing metric questions and adding several new ones to support calculating the Software Product Indicators. Section 5 discusses each of the Software Process Indicators, stating why the Methodology does not provide the right kind of data to calculate them. Section 6 summarizes the changes we recommend be made to the Methodology. Section 7 summarizes a few errors we found in the Indicator pamphlets.

2. PHILOSOPHY

This section compares and contrasts the overall purposes and philosophies of the Indicators and the Methodology.

2.1. Similarities and Differences

The Indicators and the Methodology are similar in that:

- The objective in using each of them is to improve the quality of contracted software products.
- They each use metrics—numbers calculated by formulas from project data—to give insight into the quality of the software products.

However, the Indicators and the Methodology take different approaches to achieving the objective of improved software quality.

The fundamental difference is that the Indicators are *process-related,* while the Methodology is *product-related.* The Indicators give information about the quality of the overall process of developing the software, not about any particular item under development. The Methodology gives information about the quality of each software product—each Software Requirements Specification document, Software Design Document, source code document, and so on—but gives information about the overall process only indirectly.

Sections 2.2 and 2.3 expand on the philosophies behind the Indicators and the Methodology.

2.2. Indicators' Philosophy

Because they are intended to present a high-level, overall view of the project, the Indicators consist of few metrics. The Software Management Indicators are:

- Computer Resource Utilization: percent of memory, CPU processing capacity, and I/O channel capacity used by the target computer software.
- Software Development Manpower: number of staff working on the project.
- Requirements Definition and Stability: number of software units, number of software units affected by Engineering Change Proposals (ECPs), and number of open action items.
- Software Progress—Development and Test: for each Computer Software Configuration Item (CSCI), number of units designed, number of units coded and unit tested, and number of units integrated; and for qualification testing, number of tests scheduled, number of tests passed, and number of unresolved problem reports.
- Cost/Schedule Deviations: differences between budgeted and actual cost and schedule at this point in the project.
- Software Development Tools: the date on which each software development tool is needed and the date on which it actually became available.

The data needed to compute these Software Management Indicators is project management status information that cannot be obtained by looking at the products of software development.

The Software Quality Indicators are more like traditional software product metrics. To compute each indicator, characteristics of each software product are counted, ratios are formed, and weighted sums are done to produce a single number that lies between a minimum possible and a maximum possible value. The Indicators are:

- Completeness: the degree to which requirements, design, and code documents are fully developed.

— Design Structure: the degree to which detailed designs follow software engineering principles of good structure.

— Defect Density: the ratio of defects found during inspections of design and code documents to the number of units, and the ratio of defects corrected to the number of units.

— Fault Density: the ratio of faults found during testing to the number of units, and the ratio of faults corrected to the number of units.

— Test Coverage: the fraction of the total required capabilities that has been tested, and the fraction of the software structure that has been tested.

— Test Sufficiency: estimate of the number of faults remaining in the software based on testing progress.

— Documentation: the degree to which the design and code documents are modular, descriptive, consistent, simple, expandable, and instrumented.

Again, the data needed to compute these Software Quality Indicators cannot be obtained by looking at the products of software development alone. While some of these Indicators (Completeness, Design Structure, and Documentation) depend only on product data, others (Defect Density, Fault Density, Test Coverage, and Test Sufficiency) depend on project management status data as well, such as defect data from inspections and fault data from testing. (This assertion will be justified in Section 3.) Furthermore, each Software Quality Indicator is a conglomerate number that assesses all the software documents together, not each document separately.

The Indicator pamphlets recommend that data for the Software Management Indicators and the Software Quality Indicators be collected monthly. The Indicator values for the current month and the prior twelve months should be plotted to spot trends. To determine quality levels, the plots should also show the desired values for the Indicators; large discrepancies between desired and actual values indicate areas of poor quality that may require management intervention.

Although the Indicator pamphlets recommend that reporting the Indicator data should be a contractual requirement, they do not recommend that particular *values* for the Indicators be a contractual requirement. For example, the contractor should be obligated to report the value of Fault Density (number of faults per unit) every month, but the contractor should not be obligated to achieve a Fault Density value of, say, 0.1 faults per unit before the product will be accepted. The Indicators are just that, indicators; they point out the possible existence of quality problems but are not accurate enough for their values to be used as contractual requirements.

2.3. Methodology's Philosophy

The Methodology helps to meet the goal of improved quality in the final system by focusing on each individual software document that is produced during development—each requirements, design, and code document. A large set of data is collected for each document by answering many *metric questions*. The metric questions yield numbers that are combined to give *metrics*. Selected metric scores are then combined to give *criteria*. Finally, selected criteria scores are combined to give *factors*. Factor scores lie in the range 0 to 1, with 0 representing low quality and 1 representing high quality.

The Methodology defines 13 factors, 29 criteria, 73 metrics, and 327 metric questions. Although this is a large number of metric questions, not all metric questions are answered for each document. Typically between 45 and 200 metric questions are answered for one document, depending on the kind of document and the phase of development. Furthermore, the tailoring process described below may eliminate certain factors altogether, along with the consequent metric questions.

The top-level quality factors are:

- Efficiency: How well does the software utilize resources?
- Integrity: How secure is it?
- Reliability: What confidence can be placed in what is does?
- Survivability: How well will it perform under adverse conditions?
- Usability: How easy is it to use?
- Correctness: How well does it conform to the requirements?
- Maintainability: How easy is it to repair?
- Verifiability: How easy is it to verify its performance?
- Expandability: How easy is it to expand or upgrade its capability or performance?
- Flexibility: How easy is it to change?
- Interoperability: How easy is it to interface with another system?
- Portability: How easy is it to transport?
- Reusability: How easy is it to convert for use in another application?

An acquisition manager who wants to use the Methodology on a contracted software project begins by tailoring the Methodology to match the project's characteristics. Some of the quality factors may not be important for a particular project and would be tailored out. The manager then decides on a required quality level for each remaining factor—excellent, good, or average. Tradeoff analysis is done to determine the technical and cost risks of specifying these quality levels: How technically feasible is it to achieve all these quality goals? How much more will it cost? After this analysis, the manager may decide to change some of the factors' required quality levels, to reduce the risk. The manager then puts requirements into the contract that the contractor must collect the data needed to calculate the quality factors for each document, and that the calculated quality factor values must exceed stated levels before the system will be accepted.

As the contractor produces each software document during development, the contractor (or an independent verification and validation agency) also submits the data for all the metric questions and calculates the metric, criteria, and factor scores for each document. The Methodology recommends that this be done near the end of each stage of development, in conjunction with the formal review for the acquiring agency that marks the transition to the next stage. Document scores significantly below the contractually-required levels indicate poor quality and point out areas for corrective action. The Methodology also recommends that the scores be plotted against time to spot trends.

The Methodology provides a highly detailed view of the project's quality. Quality factor scores are produced for each software document, from overall requirements and design documents to detailed design and code documents for each unit. The acquisition manager can identify the particular areas that have poor quality and that have high quality, then take selective action to improve just the areas with low quality.

The acquisition manager does not get an overall picture of the quality of the software development process directly from the Methodology, as is the case from the Indicators. However, the manager can compute a single quality score for a document as a weighted sum of its factor scores, and he or she can compute a quality score for the project as a whole as a weighted sum of all the documents' quality scores. (The Methodology guidebooks do not describe procedures for doing this.) If all the document scores are sufficiently high, then presumably the quality of the software development process is high.

2.4. Summary

Table 2-1 summarizes the Indicators' and the Methodology's philosophies.

Characteristic	Indicators' Philosophy	Methodology's Philosophy
Measurement focuses on:	Development *process*.	Development *products*.
Scope:	Overall view of the quality of the entire project.	Detailed view of the quality of each software document.
Volume:	13 Indicators (6 Management, 7 Quality).	13 factors, based on 29 criteria, based on 73 metrics.
Source of collected data:	Project management information, a few product documents.	Product documents.
When is data collected:	Monthly.	Near the end of each stage of development.
Role in contract acquisition:	Data collection is a contractual requirement.	Data collection is a contractual requirement *and* Meeting stated factor score levels is a contractual requirement.
Role in contract monitoring:	Discrepancies between actual and desired Indicator values signal possible areas for corrective action.	Low factor values signal possible areas for corrective action.

Table 2-1

3. COMBINING THE INDICATORS AND THE METHODOLOGY

3.1. Advantages of a Combination

Since both the Indicators and the Methodology have the same objective, to achieve higher quality software, and since they both use metrics to quantify quality, it seems reasonable to combine the Indicators and the Methodology. The advantages of doing so are:

— The two together give a fuller picture of a project's quality level. The Indicators give a high-level summary and give process-related information (like manpower levels and software tools status) that the Methodology does not. The Methodology gives a detailed report on each software document's quality that the Indicators do not.

— Since some of the same data feeds into both the Indicators and the Methodology, less data collection effort is needed if the Indicators and the Methodology are combined than if they are both done separately.

However, since the Indicators and the Methodology differ so widely in their philosophies, it follows that they cannot naturally be fully integrated. Also, combining them will require changing them somewhat to accommodate each other. Section 3.2 describes how we propose to combine the Indicators and the Methodology.

3.2. Approach to a Combination

The individual Indicators will be reorganized into the Software Process Indicators and the Software Product Indicators as follows:

Software Process Indicators:
- Computer Resource Utilization
- Software Development Manpower
- Requirements Definition and Stability
- Software Progress—Development and Test
- Cost/Schedule Deviations
- Software Development Tools
  (The above Software Process Indicators were originally the Software Management Indicators.)
- Defect Density
- Fault Density
- Test Coverage
- Test Sufficiency
  (The above Software Process Indicators were originally four of the seven Software Quality Indicators.)

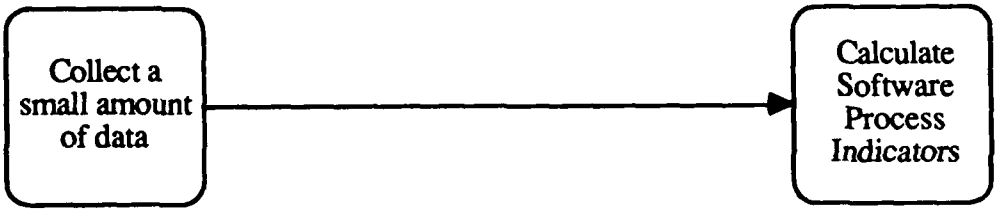
Software Product Indicators:
- Completeness
- Design Structure
- Documentation
  (The above Software Product Indicators were originally three of the seven Software Quality Indicators.)

When a system is being acquired, the acquisition manager will go through the process of establishing Methodology quality factor goals, as described in Section 2.3. Then the manager will put these requirements into the contract:

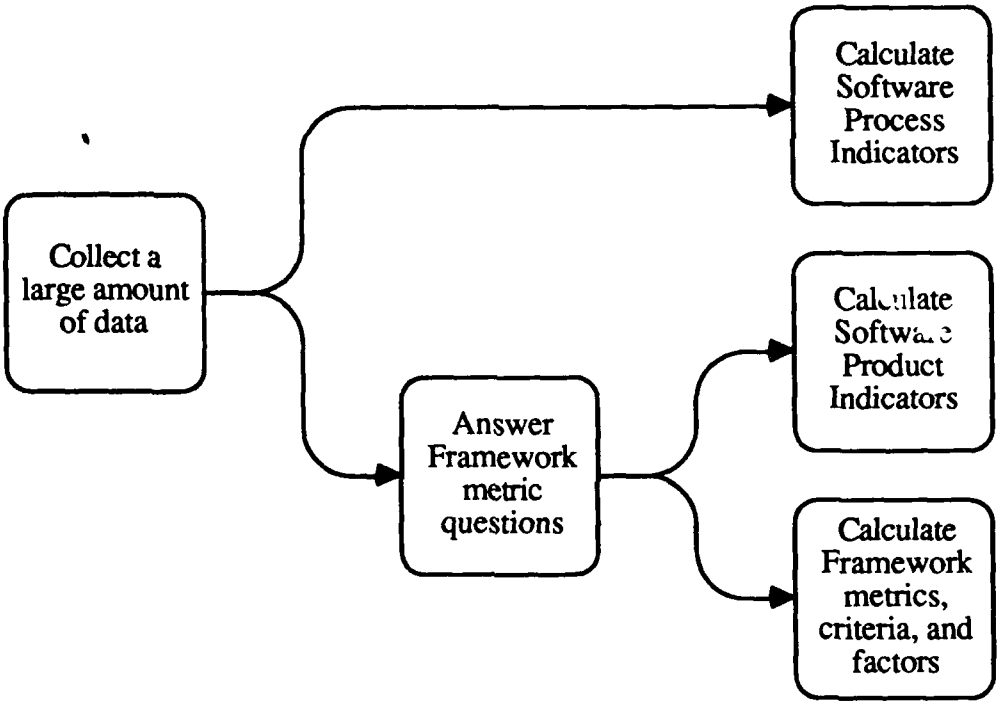
- Every month, the contractor shall supply the data needed to calculate the Software Process Indicators.
- At the end of every development phase, the contractor shall supply the data needed to calculate the Methodology quality factors for each CSCI.
- The Methodology quality factor values for each CSCI must meet or exceed certain levels before the system will be accepted.

Every month, the Software Process Indicators will be calculated, but not the Software Product Indicators. At the end of every development phase, the Methodology quality factors will be calculated, and the Software Product Indicators will be calculated from the same basic data.

Thus, in months that do not mark the end of a development phase, the data collection and analysis process will be:



In months that do mark the end of a development phase, the data collection and analysis process will be:



If the data collection and calculations are to be done automatically by a tool such as the Quality Evaluation System (QUES), then the QUES tool must operate in two different modes: one mode for determining the Software Process Indicators each month, another mode for determining the Software Process Indicators, Software Product Indicators, and Methodology scores at the end of a development phase.

3.3. Rationale For This Approach

This approach integrates the Indicators and the Methodology only to a limited extent. Only the Software Product Indicators (a subset of the former Software Quality Indicators) are integrated with the Methodology. The Software Process Indicators (the rest of the former Software Quality Indicators and all the former Software Management Indicators) are not integrated with the Methodology. There are several reasons for limiting the extent of the integration:

— *The philosophies are different.* The Software Process Indicators measure process quality, while the Methodology and the Software Product Indicators measure product quality.

— The sources for data collection are different. The Methodology's and the Software Product Indicators' data is obtained by analyzing just the software product documents. The

Software Process Indicators require management status information that is not part of any software product document.

— The amount of data collected is different. The Software Process Indicators require relatively little data; this amount of data can be collected monthly without placing an undue burden on the developer. The Methodology and the Software Product Indicators require massive amounts of data, too much to be collected every month. This data can be collected practically only once per document, at the end of the development phase in which the document was produced.

Thus, using this approach, the acquisition manager can monitor the quality of the software development process on an ongoing, monthly basis, and can monitor the quality of each software product document as it is produced.

We now turn to the details of implementing the combined Indicators and Methodology. Section 4 describes how to calculate the Software Product Indicators from the Methodology metric question data. Section 5 discusses the Software Process Indicators.

4. CALCULATING THE SOFTWARE PRODUCT INDICATORS

A different set of Methodology metric questions is answered at the end of each development phase. The Methodology organizes the metric questions for each phase into a worksheet. Here are the worksheets for each phase:

— System/Software Requirements Analysis: Metric Worksheet 0.

— Software Requirements Analysis: Metric Worksheet 1.

— Preliminary Design: Metric Worksheet 2.

— Detailed Design: Metric Worksheet 3B (filled out separately for each unit), Metric Worksheet 3A (filled out with results from all the Metric Worksheet 3Bs).

— Coding and Unit Testing: Metric Worksheet 4B (filled out separately for each unit), Metric Worksheet 4A (filled out with results from all the Metric Worksheet 4Bs).

— CSC Integration and Testing, CSCI-Level Testing, and System Integration and Testing: Selected questions from Worksheets 0, 1, 2, 3A, and 4A are answered.

The metric questions from these Methodology Metric Worksheets supply much of the data needed to calculate each of the Software Product Indicators. The next subsections describe how to do this. In some cases the requisite data is already available, and we show how to calculate both the Indicator and the metric question from that data. In other cases the data is not available in quite the format needed to support the Indicator, and we recommend how to change the applicable metric question to support both the Methodology and the Indicator. In still other cases the data for the Indicator is not available at all, and we recommend adding a metric question to the Methodology that will also support the Indicator.

In the next subsections, we will refer to "Phase 0" as an abbreviation for the System/Software Requirements Analysis phase, to "Phase 1" for the Software Requirements Analysis phase, to "Phase 2" for the Preliminary Design phase, to "Phase 3" for the Detailed Design phase, to "Phase 4" for the Coding and Unit Testing phase, and to "Testing" for the CSC Integration and Testing, CSCI-Level Testing, and System Integration and Testing phases.

4.1. Completeness

The Completeness Indicator consists of ten components that are individually calculated, then combined in a weighted sum to give an overall score. The Indicator pamphlets state that the weights depend on the development phase; that is, some components apply only during certain phases. Below we indicate how each component is defined, what its parameters are, and the phases during

which it applies. (The Indicator pamphlets do not specify the phases during which each component applies; these are our own interpretations. We have taken the definitions and the parameters directly from the Indicator pamphlets, even though sometimes the terminology is not consistent with DOD-STD-2167A.) Then we describe how to derive its parameter values from the Methodology metric questions on the worksheets for the applicable phases.

4.1.1. Completeness Component C_1: Functions Satisfactorily Defined.

Definition: $$C_1 = \frac{P_2 - P_1}{P_2}$$

Parameters: P_1 = Number of functions not adequately defined or specified.

P_2 = Total number of functions.

Applicable Phases: 0, 1, 2, 3.

Metric question CP.1(1), asks, "Are all inputs, processing, and outputs clearly and precisely defined?" This is like asking, "Does C_1 equal unity?" We recommend that metric question CP.1(1) on metric worksheets 0, 1, and 2 be changed as follows:

CP.1(1) a. How many functions in this system?

b. How many functions have clearly and precisely defined inputs, processing, and outputs?

c. Calculate b/a and enter score.

We recommend that metric question CP.1(1) on metric worksheet 3B be changed as follows:

CP.1(1) d. How many functions in this unit?

e. How many functions have clearly and precisely defined inputs, processing, and outputs?

f. Calculate b/a and enter score.

We recommend that summary metric question CP.1(1) on metric worksheet 3A be changed as follows:

CP.1(1) a. How many total functions (sum of all Worksheet 3B Question CP.1(1)d values)?

b. How many total functions have clearly and precisely defined inputs, processing, and outputs (sum of all Worksheet 3B, Question CP.1(1)e values)?

c. Calculate b/a and enter score.

Then the value (c) from the metric worksheet for the current phase (0, 1, 2, or 3A) is the desired component C_1.

4.1.2. Completeness Component C_2: Defined Data Item.

Definition: $$C_2 = \frac{P_4 - P_3}{P_4}$$

Parameters: P_3 = Number of data items not defined.

P_4 = Total number of data items.

Applicable Phases: 0, 1, 2, 3, 4.

Metric question CP.1(2) on metric worksheets 0, 1, 2, 3A, 3B, 4A, and 4B provides the requisite data. Here is the metric question:

CP.1(2) a. How many data references are identified?

b. How many identified data references are documented with regard to source, meaning, and format?

c. Calculate b/a and enter score.

The value (c) from the metric worksheet for the current phase (0, 1, 2, 3A, or 4A) is the desired component C_2.

4.1.3. Completeness Component C_3: Defined Functions Used.

Definition: $$C_3 = \frac{P_6 - P_5}{P_6}$$

Parameters: P_5 = Number of defined functions not used.

P_6 = Total number of defined functions = $P_2 - P_1$.

Applicable Phases: 0, 1.

Metric question CP.1(5) asks, "Have all defined functions (i.e., documented with regard to source, meaning, and format) been referenced?" This is like asking, "Does C_3 equal unity?" We recommend that metric question CP.1(5) on metric worksheets 0 and 1 be changed as follows:

CP.1(5) a. How many defined functions (i.e. with clearly and precisely described inputs, processing, and outputs) in this system?

b. How many defined functions are referenced?

c. Calculate b/a and enter score.

Then the value (c) the metric worksheet for the current phase (0 or 1) is the desired component C_3.

4.1.4. Completeness Component C_4: Defined Referenced Functions.

Definition: $$C_4 = \frac{P_8 - P_7}{P_8}$$

Parameters: P_7 = Number of functions referenced by defined functions but not defined.

P_8 = Total number of functions referenced by defined functions.

Applicable Phases: 0, 1.

Metric question CP.1(7) asks, "Have all referenced functions been defined (i.e., documented with precise inputs, processing, and output requirements)?" This is like asking, "Does C_4 equal unity?" We recommend that metric question CP.1(7) on metric worksheets 0 and 1 be changed as follows:

CP.1(7) a. How many functions in this system are referenced by other defined functions?

b. How many referenced functions are themselves defined (i.e. with clearly and precisely described inputs, processing, and outputs)?

c. Calculate b/a and enter score.

Then the value (c) the metric worksheet for the current phase (0 or 1) is the desired component C_4.

4.1.5. Completeness Component C_5: All Condition Options Are Used at Decision Points.

Definition: $$C_5 = \frac{P_{10} - P_9}{P_{10}}$$

Parameters: P_9 = Number of decision points not using all conditions or options.

P_{10} = Total number of decision points.

Applicable Phases: 2, 3, 4

Metric question CP.1(9) asks, "Are all conditions and alternative processing options defined for each decision point?" This is an undiscriminating question. The Completeness Indicator actually has four components (C_5, C_6, C_8, and C_9) that measure different things about decision points. We recommend that metric question CP.1(9) on metric worksheet 2 be changed as follows:

CP.1(9) a. How many decision points?

b. How many decision points have identified all possible conditions or options?

c. Calculate b/a and enter score.

We recommend that metric question CP.1(9) on metric worksheets 3B and 4B be changed as follows:

CP.1(9) d. How many decision points?

e. How many decision points have identified all possible conditions or options?

f. Calculate b/a and enter score.

We recommend that summary metric question CP.1(9) on metric worksheets 3A and 4A be changed as follows:

CP.1(9) a. How many (total) decision points (sum of all Worksheet {3B, 4B} Question CP.1(9)d values)?

b. How many (total) decision points have identified all possible conditions or options (sum of all Worksheet {3B, 4B} Question CP.1(9)e values)?

c. Calculate b/a and enter score.

Then the value (c) from the metric worksheet for the current phase (2, 3A, or 4A) is the desired component C_5.

Later we will recommend adding new metric questions to support the other Completeness Indicator components that measure decision points.

4.1.6. Completeness Component C_6: All Condition Options With Processing Are Used at Decision Points.

Definition: $$C_6 = \frac{P_{12} - P_{11}}{P_{12}}$$

Parameters: P_{11} = Number of condition options without processing.

P_{12} = Total number of condition options.

Applicable Phases: 2, 3, 4.

No metric question currently provides this data. We recommend that a new metric question, CP.1(12), be added to the existing Completeness Checklist metric, CP.1. This question should be added to metric worksheet 2:

CP.1(12) a. How many condition options in all decision points?

b. How many condition options have non-null processing associated with them?

c. Calculate b/a and enter score.

This question should be added to metric worksheets 3B and 4B:

CP.1(12) d. How many condition options in all decision points?

e. How many condition options have non-null processing associated with them?

f. Calculate b/a and enter score.

This question should be added to metric worksheets 3A and 4A:

CP.1(12) a. How many (total) condition options (sum of all Worksheet {3B, 4B} Question CP.1(12)d values)?

b. How many (total) condition options have non-null processing associated with them (sum of all Worksheet {3B, 4B} Question CP.1(12)e values)?

c. Calculate b/a and enter score.

Then the value (c) from the metric worksheet for the current phase (2, 3A, or 4A) is the desired component C_6.

4.1.7. Completeness Component C_7: All Calling Routine Parameters Agree With the Called Routine's Defined Parameters.

Definition: $$C_7 = \frac{P_{14} - P_{13}}{P_{14}}$$

Parameters: P_{13} = Number of calling routines with calling parameters that do not agree with defined parameters.

P_{14} = Total number of calling routines.

Applicable Phases: 2, 3, 4.

No metric question currently provides this data. We recommend that a new metric question, CP.1(13), be added to the existing Completeness Checklist metric, CP.1. This question should be added to metric worksheet 2:

CP.1(13) a. How many subroutine calls are made?

b. How many subroutine calls are there in which the actual parameters agree with the called subroutine's formal parameters?

c. Calculate b/a and enter score.

This question should be added to metric worksheets 3B and 4B:

CP.1(13) d. How many subroutine calls are made?

e. How many subroutine calls are there in which the actual parameters agree with the called subroutine's formal parameters?

f. Calculate b/a and enter score.

This question should be added to metric worksheets 3A and 4A:

CP.1(13) a. How many (total) subroutine calls (sum of all Worksheet {3B, 4B} Question CP.1(13)d values)?

b. How many (total) subroutine calls agree with the called subroutine's formal parameters (sum of all Worksheet {3B, 4B} Question CP.1(13)e values)?

c. Calculate b/a and enter score.

Then the value (c) from the metric worksheet for the current phase (2, 3A, or 4A) is the desired component C_7.

4.1.8. Completeness Component C_8: All Condition Options Are Set.

Definition: $$C_8 = \frac{P_{12} - P_{15}}{P_{12}}$$

Parameters: P_{15} = Number of condition options that are not set.

P_{12} = Total number of condition options (see Section 3.3.1.6).

Applicable Phases: 2, 3, 4.

The Indicator pamphlets do not describe what is meant by a condition option that is "set" or "not set." Lacking a definition, we were unable to determine whether the Methodology metric questions supply this data. This is a deficiency in the Indicator pamphlets.

4.1.9. Completeness Component C_9: All Processing Follows Set Condition Options.

Definition: $$C_9 = \frac{P_{17} - P_{16}}{P_{17}}$$

Parameters: P_{16} = Number of condition options that are set but have no processing associated with the option.

P_{17} = Number of set condition options = $P_{12} - P_{15}$.

Applicable Phases: 2, 3, 4.

The Indicator pamphlets do not describe what is meant by a condition option that is "set" but has no "associated processing." Lacking a definition, we were unable to determine whether the Methodology metric questions supply this data. This is a deficiency in the Indicator pamphlets.

4.1.10. Completeness Component C_{10}: All Data Items Have a Destination.

Definition: $$C_{10} = \frac{P_4 - P_{18}}{P_4}$$

Parameters: P_{18} = Number of data references having no destination.

P_4 = Total number of data items (see Section 3.3.1.2).

Applicable Phases: 0, 1, 2, 3, 4.

Metric question CP.1(3) on metric worksheets 0, 1, 2, 3A, and 4A provides the requisite data. Here is the metric question:

CP.1(3) a. How many data items are defined (i.e., documented with regard to source, meaning, and format)?

b. How many data items are referenced?

c. Calculate b/a and enter score.

The value (c) from the metric worksheet for the current phase (0, 1, 2, 3A, or 4A) is the desired component C_{10}.

4.1.11. Final Completeness Indicator Value.

Once the components C_1 through C_{10} have been calculated, the final Completeness Indicator value is given by one of the formulas below, depending on the development phase. By default, all applicable components for a phase are weighted equally.

Phase	Completeness Indicator Formula
0	$\frac{C_1 + C_2 + C_3 + C_4 + C_{10}}{5}$
1	$\frac{C_1 + C_2 + C_3 + C_4 + C_{10}}{5}$
2	$\frac{C_1 + C_2 + C_5 + C_6 + C_7 + C_8 + C_9 + C_{10}}{8}$
3	$\frac{C_1 + C_2 + C_5 + C_6 + C_7 + C_8 + C_9 + C_{10}}{8}$
4	$\frac{C_2 + C_5 + C_6 + C_7 + C_8 + C_9 + C_{10}}{7}$

4.2. Design Structure

The Design Structure Indicator consists of six components that are individually calculated, then combined in a weighted sum to give an overall score. The Indicator pamphlets state that this indicator should be calculated starting with the Preliminary Design Review, that is, at the end of phases 2, 3, and 4. Below we indicate how each component is defined and what its parameters are. Then we describe how to derive its parameter values from the Methodology metric questions on the worksheets for the applicable phases.

4.2.1. Design Structure Component D_1: Design Properly Organized and Structured.

Definition: $D_1 = 1$ if yes, 0 if no.

Criterion: Is the design organized top-down, bottom-up, or object-oriented as applicable, and is the design structured?

Metric question MO.1(2) on metric worksheets 3A and 4A provides the requisite data. This question asks, "Are all top-level CSCs developed according to structured design techniques?" (metric worksheet 2) "Are all CSCs developed according to structured design techniques?" (metric worksheet 3A) "Are all units coded and tested according to structural techniques?" (metric worksheet 4A) This is the essentially the same question as D_1.

4.2.2. Design Structure Component D_2: Unit Independence.

Definition: $D_2 = \frac{S_1 - S_2}{S_1}$

Parameters: S_1 = Software size expressed in terms of units.

S_2 = Number of units dependent on the source of input data (a result of prior processing or calling sequence) or destination of output data (post-processing or display).

Metric question SI.1(2) on metric worksheets 3A and 4A provides the requisite data. Here is the metric question:

SI.1(2) a. How many applicable units?

b. How many units with answer of Y [to the question, Is the unit independent of the source of the input and the destination of the output]?

c. Calculate b/a and enter score.

Metric question SI.1(2) does not currently appear on metric worksheet 2. To support the Design Structure Indicator during phase 2, we recommend that it be added, as follows:

SI.1(2) a. How many CSCs in the software?

b. How many CSCs are independent of the source of the input and the destination of the output?

c. Calculate b/a and enter score.

The value (c) from the metric worksheet for the current phase (2, 3A, or 4A) is the desired component D_2.

4.2.3. Design Structure Component D_3: Units Not Dependent On Prior Processing.

Definition: $$D_3 = \frac{S_1 - S_3}{S_1}$$

Parameters: S_1 = Software size expressed in terms of units.

S_3 = Number of units dependent on prior processing.

Metric question SI.1(3) on metric worksheets 3A and 4A provides the requisite data. Here is the metric question:

SI.1(3) a. How many applicable units?

b. How many units with answer of Y [to the question, Is the unit independent of the knowledge of prior processing]?

c. Calculate b/a and enter score.

Metric question SI.1(3) does not currently appear on metric worksheet 2. To support the Design Structure Indicator during phase 2, we recommend that it be added, as follows:

SI.1(3) a. How many CSCs in the software?

b. How many CSCs are independent of the knowledge of prior processing?

c. Calculate b/a and enter score.

The value (c) from the metric worksheet for the current phase (2, 3A, or 4A) is the desired component D_3.

4.2.4. Design Structure Component D_4: Data Base Size.

Definition: $$D_4 = \frac{S_4 - S_5}{S_4}$$

Parameters: S_4 = Number of data base items. This is the same as P_4 (see Section 3.3.1.2).

S_5 = Total number of unique data base items, including local and global data bases.

First, this component is defined incorrectly. The ideal situation is when every data base item is unique; when there are no instances of two or more items storing the same data. In other words, S_4 ideally should equal S_5; in a poor design, S_4 will be greater than S_5. But with the

above definition of D_4, if S_4 equals S_5, D_4 equals 0, not 1; and the larger (poorer) S_4 becomes, the closer to 1 (better) D_4 gets. This is backwards. The correct definition is $D_4 = S_5/S_4$.

No metric question currently provides this data. In the Methodology, this Indicator component fits most closely under the Simplicity criterion, yet none of the existing Simplicity metrics deal with data base items. We therefore recommend that a new metric, Data Simplicity, SI.7, be added to the Simplicity criterion. We recommend that a new metric question, SI.7(1), be added to metric worksheets 2, 3A, and 4A:

SI.7(1) a. How many total data base items?

b. How many unique data base items? (If the same value is stored in more than one data base item, all of those data base items count as one unique data base item.)

c. Calculate b/a and enter score.

The value (c) from the metric worksheet for the current phase (2, 3A, or 4A) is the desired component D_4.

4.2.5. Design Structure Component D_5: Data Base Compartmentalization.

Definition: $$D_5 = \frac{S_4 - S_6}{S_4}$$

Parameters: S_4 = Number of data base items. This is the same as P_4 (see Section 3.3.1.2).

S_6 = Number of data base segments.

The Indicator pamphlets do not define what a "data base segment" is; this is a deficiency in the Pamphlets. We interpret a "data base segment" to be "the set of data items referenced by a single access to the data base." That is, if a single subroutine call (or whatever is used to access the data base) yields more than one data base item, those items are considered to be in the same data base segment; they cannot be accessed separately.

If that is what a data base segment is, then this component is defined incorrectly. The ideal situation is when every data base item is in a separate data base segment; when each item can be accessed independently of all the others. In other words, S_6 ideally should equal S_4; in a poor design, S_6 will be less than S_4. But with the above definition of D_5, if S_6 equals S_4, D_5 equals 0, not 1; and the smaller (poorer) S_6 becomes, the closer to 1 (better) D_5 gets. This is backwards. The correct definition is $D_5 = S_6/S_4$.

No metric question currently provides this data. In the Methodology, this Indicator component fits most closely under the Simplicity criterion, under the new metric Data Simplicity, SI.7. We recommend that a new metric question, SI.7(2), be added to metric worksheets 2, 3A, and 4A:

SI.7(2) a. How many total data base items?

b. How many total data base segments? (If a single access to the data base yields more than one data base item, all of those data base items count as one data base segment.)

c. Calculate b/a and enter score.

The value (c) from the metric worksheet for the current phase (2, 3A, or 4A) is the desired component D_5.

4.2.6. Design Structure Component D_6: Unit Single Entrance/Single Exit.

Definition: $$D_6 = \frac{S_7}{S_1}$$

Parameters: S_1 = Software size expressed in terms of units.

S_7 = Number of units with a single entrance/single exit. Branch on error detection is not considered as a multiple exit condition.

Metric question SI.1(5) on metric worksheets 3B and 4B approaches this metric in a different way. Here is the metric question:

SI.1(5) d. How many entrances into the unit?

e. How many exits from the unit?

f. Calculate $\frac{1}{2d} + \frac{1}{2e}$ and enter score.

The ideal situation is when a unit has just one entrance and just one exit. Both the Indicator component and the Methodology metric question would give this unit a score of 1. The difference comes when a unit deviates from this ideal. Say a unit has two entrances and one exit. The Methodology metric question would give this unit a score of 3/4, but the Indicator component would give it a score of 0. The Methodology metric question gradually increases the penalty the more entrances and exits a unit has. The Indicator component assesses the maximum possible penalty as soon as a unit has more than one entrance or exit. The latter approach is more in accord with current software engineering practice.

We therefore recommend that metric question SI.1(5) on metric worksheets 3B and 4B be changed as follows:

SI.1(5) *d. Does the unit have just one entrance and just one exit?* Y/N

We recommend that metric question SI.1(5) on metric worksheets 3A and 4A be changed as follows:

SI.1(5) a. How many applicable units (answer of Y or N on {3B, 4B})?

b. How many units with answer of Y (see {3B, 4B})?

c. Calculate b/a and enter score.

Metric question SI.1(5) does not currently appear on metric worksheet 2. To support the Design Structure Indicator during phase 2, we recommend that it be added, as follows:

SI.1(5) a. How many CSCs in the software?

b. How many CSCs have just one entrance and just one exit?

c. Calculate b/a and enter score.

The value (c) from the metric worksheet for the current phase (2, 3A, or 4A) is the desired component D_6.

4.2.7. Final Design Structure Indicator Value.

Once the components D_1 through D_6 have been calculated, the final Design Structure Indicator value is given by the formula below. By default, all components are weighted equally.

$$\text{Design Structure} = \frac{D_1 + D_2 + D_3 + D_4 + D_5 + D_6}{6}$$

4.3. Documentation

The Documentation Indicator is based on subjective evaluations of the quality of the system's documentation (all documents except source code) and of the quality of the system's source code. Note that this does not refer to the quality of the system's design itself or the quality of the system's code itself, but rather to the quality of the documentation that describes the design and the quality of the listings that carry the source code.

First, the set of all non-source-code documents is rated for the following characteristics on a scale from 1 to 6, where 1 stands for the lowest possible quality and 6 the highest:

D_1 = Modularity of the documentation (1 to 6).
D_2 = Descriptiveness of the documentation (1 to 6).
D_3 = Consistency of the documentation (1 to 6).
D_4 = Simplicity of the documentation (1 to 6).
D_5 = Expandability of the documentation (1 to 6).
D_6 = Testability or instrumentation characteristics of the documentation (1 to 6).

Then the set of all source listings is rated for the same characteristics on a scale from 1 to 6:

S_1 = Modularity of the source listings (1 to 6).
S_2 = Descriptiveness of the source listings (1 to 6).
S_3 = Consistency of the source listings (1 to 6).
S_4 = Simplicity of the source listings (1 to 6).
S_5 = Expandability of the source listings (1 to 6).
S_6 = Testability or instrumentation characteristics of the source listings (1 to 6).

The Documentation Indicator is calculated as a weighted sum of these twelve numbers and normalized to lie in the range 0 to 1. Assuming all the weights are the same, the formula is:

$$\text{Documentation} = \frac{D_1+D_2+D_3+D_4+D_5+D_6+S_1+S_2+S_3+S_4+S_5+S_6}{72}$$

Incidentally, paragraph 2-66 of AFSC Pamphlet 800-14, the Software Quality Indicators Pamphlet, states that the Documentation Indicator has a range of values between 1 and 6. This is inconsistent with the formula given in paragraph 2-62 (and reproduced above), which yields a range of values between 0 and 1.

The Methodology has a metric DO.2, Well-Structured Documentation, that asks questions similar to those above. But while the Documentation Indicator has a range of ratings from 1 to 6, the Methodology metric questions have yes-no, all-or-nothing answers. For example, here is metric question DO.2(1) from metric worksheet 0:

DO.2(1) Is all the documentation structured and written clearly and simply such that procedures, functions, algorithms, etc. can be easily understood? Y N N/A

Both to support the Indicators and to obtain a more precise rating of documentation quality, the DO.2 metric questions should be changed to ask for ratings on a scale of 1 to 6. Furthermore, the DO.2 metric questions do not cover all six characteristics of the Documentation Indicator, requiring new metric questions to be added. With these changes, the Documentation Indicator components D_1 through D_6 and S_1 through S_6 are obtained directly from the appropriate Methodology metric questions, and are then combined by the above formula to yield the Indicator score.

4.3.1. Changes to Existing Metric Questions for Documentation Indicators.

We recommend changing metric question DO.2(1) on metric worksheets 0, 1, 2, and 3A to the following. This yields the Documentation Indicator component D_4 (simplicity of the documentation).

DO.2(1) a. Are all the documents structured and written clearly and simply such that procedures, functions, algorithms, etc. can be easily understood? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend changing metric question DO.2(1) on metric worksheet 4A to the following. This yields the Documentation Indicator component D_4 (simplicity of the documentation) during phase 4.

DO.2(1) a. Are all the documents other than source listings structured and written clearly and simply such that procedures, functions, algorithms, etc. can be easily understood? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend changing metric question DO.2(2) on metric worksheets 0, 1, 2, and 3A to the following. This metric question is not part of the Documentation Indicator.

DO.2(2) a. Do the documents clearly depict control and data flow (e.g., graphic portrayal with accompanying explanations or PDL)? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend changing metric question DO.2(3) on metric worksheets 0, 1, 2, 3A, and 4A to the following. This metric question is not part of the Documentation Indicator.

DO.2(3) a. Does all documentation contain an indexing scheme which facilitates quickly locating and accessing various information in the document (e.g. hierarchically structured table of contents, inserted tabs)? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend changing metric question DO.2(4) on metric worksheets 0, 1, 2, and 3A to the following. This yields the Documentation Indicator component D_1 (modularity of the documentation).

DO.2(4) a. Do the documents have separate volumes or separations within a single volume based on system functions or software functions? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend changing metric question DO.2(4) on metric worksheet 4A to the following. This yields the Documentation Indicator component D_1 (modularity of the documentation) during phase 4.

DO.2(4) a. Do the documents other than source listings have separate volumes or separations within a single volume based on system functions or software functions? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend changing metric question DO.2(5) on metric worksheets 0, 1, and 2 to the following. This metric question is not part of the Documentation Indicator.

DO.2(5) a. Does the documentation completely characterize the operational capabilities of the software (e.g., identify all the performance parameters and limitations)? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend changing metric question DO.2(6) on metric worksheets 0 and 1 to the following. This yields the Documentation Indicator component D_2 (descriptiveness of the documentation) during phases 0 and 1.

DO.2(6) a. Does the documentation contain comprehensive descriptions of all system and software functions including functional processing, functional algorithms, and functional interfaces? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend changing metric question DO.2(7) on metric worksheets 2 and 3A to the following. This yields the Documentation Indicator component D_2 (descriptiveness of the documentation) during phases 2 and 3.

DO.2(7) a. Does the documentation contain comprehensive descriptions of all algorithms used and their limitations, including inputs, outputs, and required precision? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(7) to metric worksheet 4A, as follows. This yields the Documentation Indicator component D_2 (descriptiveness of the documentation) during phase 4.

DO.2(7) a. Do the documents other than source listings contain comprehensive descriptions of all algorithms used and their limitations, including inputs, outputs, and required precision? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

4.3.2. New Metric Questions for Documentation Indicators.

At this point we have exhausted the existing DO.2 metric questions and have covered the Documentation Indicator components modularity, descriptiveness, and simplicity. The Documentation Indicator components consistency, expandability, and testability or instrumentation characteristics have not been covered. New DO.2 metric questions must be added.

We recommend adding metric question DO.2(9) to metric worksheets 0, 1, 2, and 3A, as follows. This yields the Documentation Indicator component D_3 (consistency of the documentation).

DO.2(9) a. Are each of the documents internally consistent, and are all the documents consistent with each other? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(9) to metric worksheet 4A, as follows. This yields the Documentation Indicator component D_3 (consistency of the documentation) during phase 4.

DO.2(9) a. Are each of the documents other than source listings internally consistent, and are all the documents consistent with each other? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(10) to metric worksheets 0, 1, 2, and 3A, as follows. This yields the Documentation Indicator component D_5 (expandability of the documentation).

DO.2(10) a. Are all of the documents able to be easily expanded to incorporate new material? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(10) to metric worksheet 4A, as follows. This yields the Documentation Indicator component D_5 (expandability of the documentation) during phase 4.

DO.2(10) a. Are all of the documents other than source listings able to be easily expanded to incorporate new material? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(11) to metric worksheets 0, 1, 2, and 3A, as follows. This yields the Documentation Indicator component D_6 (testability of the documentation).

DO.2(11) a. Are all of the documents written so that the software can be easily tested? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(11) to metric worksheet 4A, as follows. This yields the Documentation Indicator component D_6 (testability of the documentation) during phase 4.

DO.2(11) a. Are all of the documents other than source listings written so that the software can be easily tested? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

4.3.3. New Metric Questions for Source Listing Indicators.

At this point we have covered the Documentation Indicator components D_1 through D_6. None of the Documentation Indicator components dealing with source listings during phase 4 (S_1 through S_6) has been covered. New DO.2 metric questions must be added to metric worksheet 4A.

We recommend adding metric question DO.2(12) on metric worksheet 4A, as follows. This yields the Documentation Indicator component S_4 (simplicity of the source listings) during phase 4.

DO.2(12) a. Are all the source listings structured and written clearly and simply such that procedures, functions, algorithms, etc. can be easily understood? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(13) on metric worksheet 4A, as follows. This yields the Documentation Indicator component S_1 (modularity of the source listings) during phase 4.

DO.2(13) a. Do the source listings have separate volumes or separations within a single volume based on system functions or software functions? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(14) to metric worksheet 4A, as follows. This yields the Documentation Indicator component S_2 (descriptiveness of the source listings) during phase 4.

DO.2(14) a. Do the source listings contain comprehensive descriptions of the algorithms used for each module and their limitations, including inputs, outputs, and required precision? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(15) to metric worksheet 4A, as follows. This yields the Documentation Indicator component S_3 (consistency of the source listings) during phase 4.

DO.2(15) a. Are each of the source listings internally consistent, and are all the source listings consistent with each other? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(16) to metric worksheet 4A, as follows. This yields the Documentation Indicator component S_5 (expandability of the source listings) during phase 4.

DO.2(16) a. Are all of the source listings able to be easily expanded to incorporate new material? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

We recommend adding metric question DO.2(17) to metric worksheet 4A, as follows. This yields the Documentation Indicator component D_6 (testability of the documentation) during phase 4.

DO.2(17) a. Are all of the source listings written so that the software can be easily tested? Rate on a scale of 1 to 6 (1=poor, 6=excellent).

b. Calculate a/6 and enter score.

4.3.4. Final Documentation Indicator Value.

During phases 0, 1, 2, and 3, once the components D_1 through D_6 have been calculated, the final Documentation Indicator value is given by the formula below. By default, all components are weighted equally.

$$\text{Documentation} = \frac{D_1 + D_2 + D_3 + D_4 + D_5 + D_6}{6}$$

During phase4, once the components D_1 through D_6 and S_1 through S_6 have been calculated, the final Documentation Indicator value is given by the formula below. By default, all components are weighted equally.

$$\text{Documentation} = \frac{D_1 + D_2 + D_3 + D_4 + D_5 + D_6 + S_1 + S_2 + S_3 + S_4 + S_5 + S_6}{12}$$

5. CALCULATING THE SOFTWARE PROCESS INDICATORS

Since the Software Process Indicators are calculated primarily from process-oriented data, they are not integrated with the Methodology, as the Software Product Indicators are. Data for the Software Process Indicators is collected every month. At the end of each development phase, one set of data for the Software Process Indicators is collected, and another set of data for the Methodology metric questions and the Software Product Indicators is collected. Although these two sets of data are largely disjoint, there are a few data items that overlap.

The sections below discuss areas of overlap between the Software Process Indicators data and the Methodology metric question data. However, the amount of overlap is not sufficient to justify integrating the Software Process Indicators into the Methodology, as was possible with the Software Product Indicators.

Some of the Software Process Indicators require data that is not collected by the Methodology metric questions. This data must still be collected by some other means, such as the QUES automated tool. However it is collected, this data is not part of the Methodology, which is devoted to software product quality, not process quality.

5.1. Computer Resource Utilization

The Computer Resource Utilization Indicator shows the amount allocated and the amount used for three resources: memory, CPU, and I/O. Starting with metric worksheet 2, the Methodology collects the same data to compute the Augmentability criterion, AT. (This starts during phase 2 because during phases 0 and 1, the software has not yet been designed in the detail required to

estimate or measure resource amounts used.) During phase 2 and after, in the months when Methodology data is collected, the data for this Indicator can be obtained from these Methodology metric questions:

- Primary memory, amount allocated: AT.1(2)a.
- Primary memory, amount used: AT.1(2)b.
- Auxiliary storage space, amount allocated: AT.1(3)a.
- Auxiliary storage space, amount used: AT.1(3)b.
- CPU processing time, amount allocated: AT.2(3)a.
- CPU processing time, amount used: AT.2(3)b.
- I/O channel time, amount allocated: AT.3(1)a.
- I/O channel time, amount used: AT.3(1)b.
- Communication channel time, amount allocated: AT.3(2)a.
- Communication channel time, amount used: AT.3(2)b.

If the Computer Resource Utilization Indicator data can be obtained from the Methodology metric question data, why is this Indicator not included with the Software Product Indicators and integrated with the Methodology? Because this Indicator is a critical measure of the developing system's viability and needs to be sampled more often than at the end of each development phase.

5.2. Software Development Manpower

The Software Development Manpower Indicator shows actual and planned staffing for the project. This is project status information that is not collected by the Methodology metric questions (but might be collected separately by QUES to calculate the Software Process Indicators).

5.3. Requirements Definition and Stability

The Requirements Definition and Stability Indicator consists of six data items:

a. Total number of software requirements.

Although some of the Methodology metric questions ask the number of *functions*, none ask the number of *requirements*. The number of requirements must be counted separately from the requirements document.

b. Number of software requirements that are not traceable (up and down through documentation).

Although there are Methodology metric questions under a Traceability (TC) criterion, these questions merely ask "Is there a traceability table?" rather than "How many software requirements are not traceable?" The latter question must be answered by correlating the information in all the traceability tables with the original list of requirements.

c. Number of software requirements that are not testable.

None of the Methodology metric questions asks about testability of requirements. This must be determined separately.

d. Total size of the software, in units.

During Phases 0, 1, and 2, none of the Methodology metric questions asks about the number of units, although the number is easily counted. During phases 3 and 4, many of the Methodology metric questions calculate the ratio of some quantity to the total number of units. Either way the number of units is readily available.

e. Software units affected by ECPs.

Although Methodology metric question CP.1(11) asks about the total number of software problem reports to date and the number of closed (resolved) software problem reports to date, no question asks about the number of units that have been affected by software problem reports (that is, Engineering Change Proposals or ECPs). This must be determined separately.

f. Open action items.

In the months when Methodology data is collected, the number of open software problem reports (that is, action items) is given by the difference between metric question CP.1(11)a (cumulative number of software problem reports recorded) and CP.1(11)b (cumulative number of software problem reports corrected). In the other months, the number of open software problem reports must be counted separately.

5.4. Software Progress—Development and Test

The Software Progress—Development and Test Indicator shows percentage of total units fully designed, percentage of total units fully coded and unit tested, and percentage of total units fully integrated. This is project status information that is not collected by the Methodology metric questions (but might be collected separately by QUES to calculate the Software Process Indicators).

5.5. Cost/Schedule Deviations

The Cost/Schedule Deviations Indicator shows cost variance, schedule variance, cost performance index, schedule performance index, estimate at completion, and variance at completion. This is project status information that is not collected by the Methodology metric questions (but might be collected separately by QUES to calculate the Software Process Indicators).

5.6. Software Development Tools

The Software Development Tools Indicator shows, for each tool, the date on which it is required and the date on which it was delivered. This is project status information that is not collected by the Methodology metric questions (but might be collected separately by QUES to calculate the Software Process Indicators).

5.7. Defect Density

The Defect Density Indicator consists of two metrics (if the system has more than one CSCI, the metrics are calculated separately for each CSCI):

— Cumulative number of defects encountered divided by the total number of units.

— Cumulative number of defects corrected divided by the total number of units.

Defects are defined to be requirements errors, design errors, and coding errors found only during inspections and walkthroughs (not during later testing).

Although Methodology metric question CP.1(11) on metric worksheets 0, 1, 2, 3A, and 4A asks the cumulative number of software problem reports recorded and corrected, it does not distinguish which of these software problem reports are defects by the above definition. The defects must be counted separately.

5.8. Fault Density

The Fault Density Indicator consists of two metrics (if the system has more than one CSCI, the metrics are calculated separately for each CSCI):

— Cumulative number of faults encountered divided by the total number of units.

— Cumulative number of faults corrected divided by the total number of units.

Faults are defined to be software problems found only during testing (not during earlier inspections or walkthroughs).

Although Methodology metric question CP.1(11) on metric worksheets 0, 1, 2, 3A, and 4A asks the cumulative number of software problem reports recorded and corrected, it does not distinguish which of these software problem reports are faults by the above definition. The faults must be counted separately.

5.9. Test Coverage

The Test Coverage Indicator shows the degree to which testing has been completed, using this formula:

$$\text{Test Coverage} = \frac{\text{Number of Implemented Capabilities Tested}}{\text{Total Required Capabilities}} \times \frac{\text{Software Structure Tested}}{\text{Total Software Structure}} \times 100\%$$

This is project status information that is not collected by the Methodology metric questions (but might be collected separately by QUES to calculate the Software Process Indicators).

5.10. Test Sufficiency

The Test Sufficiency Indicator is an estimate of the number of faults remaining in the software, based on the total number of faults predicted in the software (derived from a software reliability model), the number of faults found before testing began, the number of faults found during testing, and the fraction of units integrated. This is project status information that is not collected by the Methodology metric questions (but might be collected separately by QUES to calculate the Software Process Indicators).

6. SUMMARY OF ERRORS IN THE INDICATORS

We found two errors in the Design Structure Software Quality Indicator in AFSC Pamphlet 800–14. The formula for the Data Base Size component, D_4, is defined incorrectly; see Section 4.2.4 above. The formula for the Data Base Compartmentalization component, D_5, is defined incorrectly; see Section 4.2.5 above.

We found one inconsistency in the Documentation Software Quality Indicator in AFSC Pamphlet 800–14. The formula for this Indicator yields a number in the range 0 to 1, like all the other Indicators. Yet paragraph 2-66 states that this Indicator ranges from 1 to 6. See Section 4.3 above.

We found four areas of deficiency in AFSC Pamphlet 800–14. There is no definition of what a "set condition option" means in the Completeness Indicator component C_8. There is no definition of what a "set condition option with processing" means in the Completeness Indicator component C_9. There is no definition of what a "data base segment" means in the Design Structure Indicator component D_5. Having been written prior to DOD-STD-2167A, the terminology used in the Indicator pamphlets is sometimes inconsistent with DOD-STD-2167A.

Although we point these out, we have not been tasked to revise the Indicator pamphlets in the current contract.

7. SUMMARY OF RECOMMENDED METHODOLOGY CHANGES

The tables on the next page list the Methodology metric questions that we recommend be changed and be added. The "See" column gives the section number in this document where the details of our recommendations may be found.

CHANGES

Worksheet	Question	See
0	CP.1(1)	4.1.1
0	CP.1(5)	4.1.3
0	CP.1(7)	4.1.4
0	DO.2(1)	4.3.1
0	DO.2(2)	4.3.1
0	DO.2(3)	4.3.1
0	DO.2(4)	4.3.1
0	DO.2(5)	4.3.1
0	DO.2(6)	4.3.1
1	CP.1(1)	4.1.1
1	CP.1(5)	4.1.3
1	CP.1(7)	4.1.4
1	DO.2(1)	4.3.1
1	DO.2(2)	4.3.1
1	DO.2(3)	4.3.1
1	DO.2(4)	4.3.1
1	DO.2(5)	4.3.1
1	DO.2(6)	4.3.1
2	CP.1(1)	4.1.1
2	CP.1(9)	4.1.5
2	DO.2(1)	4.3.1
2	DO.2(2)	4.3.1
2	DO.2(3)	4.3.1
2	DO.2(4)	4.3.1
2	DO.2(5)	4.3.1
2	DO.2(7)	4.3.1
3A	CP.1(9)	4.1.5
3A	DO.2(1)	4.3.1
3A	DO.2(2)	4.3.1
3A	DO.2(3)	4.3.1
3A	DO.2(4)	4.3.1
3A	DO.2(7)	4.3.1
3A	SI.1(5)	4.2.6
3B	CP.1(9)	4.1.5
3B	SI.1(5)	4.2.6
4A	CP.1(9)	4.1.5
4A	DO.2(1)	4.3.1
4A	DO.2(4)	4.3.1
4A	SI.1(5)	4.2.6
4B	CP.1(9)	4.1.5
4B	SI.1(5)	4.2.6

ADDITIONS

Worksheet	Question	See
0	DO.2(9)	4.3.2
0	DO.2(10)	4.3.2
0	DO.2(11)	4.3.2
1	DO.2(9)	4.3.2
1	DO.2(10)	4.3.2
1	DO.2(11)	4.3.2
2	CP.1(12)	4.1.6
2	CP.1(13)	4.1.7
2	DO.2(9)	4.3.2
2	DO.2(10)	4.3.2
2	DO.2(11)	4.3.2
2	SI.1(2)	4.2.2
2	SI.1(3)	4.3.2
2	SI.1(5)	4.2.6
2	SI.7(1)	4.2.4
2	SI.7(2)	4.2.5
3A	CP.1(12)	4.1.6
3A	CP.1(13)	4.1.7
3A	DO.2(9)	4.3.2
3A	DO.2(10)	4.3.2
3A	DO.2(11)	4.3.2
3A	SI.7(1)	4.2.4
3A	SI.7(2)	4.2.5
3B	CP.1(12)	4.1.6
3B	CP.1(13)	4.1.7
4A	CP.1(12)	4.1.6
4A	CP.1(13)	4.1.7
4A	DO.2(7)	4.3.1
4A	DO.2(9)	4.3.2
4A	DO.2(10)	4.3.2
4A	DO.2(11)	4.3.2
4A	DO.2(12)	4.3.3
4A	DO.2(13)	4.3.3
4A	DO.2(14)	4.3.3
4A	DO.2(15)	4.3.3
4A	DO.2(16)	4.3.3
4A	DO.2(17)	4.3.3
4A	SI.7(1)	4.2.4
4A	SI.7(2)	4.2.5
4B	CP.1(12)	4.1.6
4B	CP.1(13)	4.1.7

8. REFERENCES

[1] AFSC Pamphlet 800-43, Software Management Indicators, Jan. 1986

[2] AFSC Pamphelt 800-14, Software Quality Indicators, Jan., 1987

[3] T.P. Bowen, G. B. Wigle, and J. T. Tsai "Specification of Software Quality Attributes," RADC-TR-85-37, Rome Air Development Center, Feb., 1985

[4] DOD-STD-2167A. *Military Standard Defense System Software Development*, Feb.,1988

SECTION V

STUDY of METRIC QUESTION TRACEABILITY

1. INTRODUCTION

The Rome Air Development Center has sponsored ongoing research into a software quality specification and measurement methodology, hereafter referred to as the "Methodology" The Methodology is documented in a three volume set of guidebooks:

— *Specification of Software Quality Attributes*, RADC-TR-85-37, Vols. I–III. [1]

This paper reports on our study of the traceability of the Methodology's metric questions across the sequential phases of the software development lifecycle.

2. TRACEABILITY CONCERNS

Appendix A of Volume III contains a cross reference listing showing all the metric worksheets on which each metric question appears. For example, metric question CP.1(2), part of the Completeness criterion, is asked during the System/Software Requirements Analysis phase on Metric Worksheet 0; during the Software Requirements Analysis phase on Metric Worksheet 1; during the Preliminary Design phase on Metric Worksheet 2; during the Detailed Design phase on Metric Worksheets 3A and 3B; and during the Coding and Unit Testing phase on Metric Worksheets 4A and 4B. This question is not asked during the CSC Integration and Testing, the CSCI-Level Testing, or the System Integration and Testing phases.

It can be seen that metric question CP.1(2) is asked repeatedly during each sequential development phase. Thus, the quality concern this question addresses is tracked from phase to phase to ensure that it is addressed throughout the development process. We say that *traceability* is provided for this quality concern across the development process.

Studying Appendix A, one quickly discovers that there seem to be traceability gaps in certain metric questions. For example, the AM.1(1) metric question is asked during the System/Software Requirements Analysis phase on Metric Worksheet 0 and during the Software Requirements Analysis phase on Metric Worksheet 1. Subsequently, that question appears on no other worksheet and is not asked during the Preliminary Design, Detailed Design, Coding and Unit Testing, and CSC Integration and Testing phases. After this gap, the question reappears during the CSCI-Level Testing phase on Metric Worksheet 1 and the System Integration and Testing phase on Metric Worksheet 0. Many other examples are apparent in Appendix A.

As part of this contract we were tasked with answering the following questions:

— Is it true that because of these traceability gaps, important software quality concerns will not be addressed during certain development phases?

— Should new metric questions be written to fill the gaps?

— If so, what are the metric questions that should be added?

We studied each metric question gap and concluded that in all cases there are valid reasons for the gap's existence. Traceability of software quality concerns is not lost due to these gaps. No new metric questions need to be added. Section 3 gives the rationale for these conclusions.

3. RATIONALE FOR METRIC QUESTION GAPS

3.1. Software Quality Addressed By Technical Reviews

Some metric questions are asked on Metric Worksheets 0, 1, and 2 during the System/Software Requirements Analysis, Software Requirements Analysis, and Preliminary Design phases. Then comes a gap during the Detailed Design and Coding and Unit Testing phases. The metric question picks up again on Metric Worksheet 2 during the CSC Integration and Testing and later testing phases.

One example is metric question AC.1(7) on Metric Worksheet 2: "Do the numerical techniques used in implementing applicable functions (e.g., mission-critical functions) provide enough precision to support accuracy objectives?"

The metric questions that fall in this category are AC.1(7), AM.1(1), AM.1(2), AM.1(4), AM.4(1), AM.5(1), AM.6(1), AM.6(2), AM.6(3), AM.6(4), AM.7(1), AM.7(2), AM.7(3), AT.4(1), AU.2(2),

These questions are asked as part of the Methodology up to the Preliminary Design phase in order to ensure that various quality concerns are addressed during the early stages of development. As a result, requirements are written into the requirements documents and modules are placed in the software design to address the quality concerns and achieve a high level of software quality. The Methodology metric questions make sure that these elements have been written into the requirements and design documents. After that point, however, the normal product technical review process (including internal contractor inspections and reviews as well as the Software Specification Review, Preliminary Design Review, and Critical Design Review) will ensure that everything in the software requirements and design documents gets properly implemented, including the modules that were affected by the quality concerns. There is no need for the Methodology to duplicate what the normal product technical review process will anyway ensure. When the project reaches the testing stages, the metric question reappears to ensure that the product will be tested for compliance with the quality requirements. Hence, a gap appears between the Preliminary Design and the CSC Integration and Testing phases for the metric question.

Consider metric question AC.1(7) in particular. Asking this question during the Preliminary Design phase ensures that all modules in the software are designed in such a way as to provide the required numerical accuracy. The normal product technical review process will ensure that each module's detailed design and code are correct with respect to the design specification—that each module's detailed design and code provides the required numerical accuracy. There is no need to ask the Methodology metric question at this point to obtain the necessary software quality.

3.2. Software Quality Addressed By Implementation Standards

Some metric questions are asked on Metric Worksheets 0 and 1 during the System/Software Requirements Analysis and Software Requirements Analysis phases. Then comes a gap during the Preliminary Design phase. The metric question picks up again on Metric Worksheets 3A, 3B, 4A, and 4B during the Detailed Design and Coding and Unit Testing phases.

One example is metric question AM.1(3) on Metric Worksheet 1: "Is there a standard for handling recognized errors such that all error conditions are passed to the calling function or software element?" On Metric Worksheet 3B, this question is stated as "When an error condition is detected, is resolution of the error determined by the calling unit?" Metric Worksheet 3A asks for the proportion of units for which this is true.

The metric questions that fall in this category are AM.1(3), AM.3(2), AM.3(3), AM.3(4), AP.2(2), AP.2(4), AP.3(1), AP.4(1),

These questions are asked during the Software Requirements Analysis phase to ensure that various implementation standards are set up early in the software development lifecycle. These standards are implementation-related, not design-related, and so the metric question is irrelevant during the Preliminary Design phase. When the project gets to the implementation phases of Detailed Design and Coding and Unit Testing, the metric questions are asked to ensure that the product complies with the implementation standards previously established. Hence, a gap appears between the Software Requirements Analysis and the Detailed Design phases for the metric question.

3.3. Hardware Issues

Some metric questions ask about hardware characteristics that are required for obtaining a high-quality system. Such questions are applicable only in the development phases where hardware and software are both being considered, that is, the System/Software Requirements Analysis, Software Requirements Analysis, CSCI-Level Testing, and System Integration and Testing phases. Hence, a gap appears between the Software Requirements Analysis and the CSCI-Level Testing phases for the metric question.

One example is metric question AU.2(1) on Metric Worksheet 1: "Does each operational CPU/system have a separate power source?"

The metric questions that fall in this category are AU.2(1),

3.4. Miscellaneous Metric Question Gaps

Metric questions AP.5(2) and AP.5(3) are asked during the Preliminary Design phase on Metric Worksheet 2, and are not asked again until the CSC Integration and Testing phase on Metric Worksheet 2. Metric question AP.5(2) asks for the proportion of algorithms that have been verified with respect to their requirements. Metric question AP.5(3) asks for the proportion of algorithms that have test data available which reflects results of algorithm verification. Since verification of *algorithms* is a *design* concern, not an implementation concern, there is no need to ask these metric questions during the Detailed Design and Coding and Unit Testing stages.

4. REFERENCES

[1] T.P. Bowen, G. B. Wigle, and J. T. Tsai "Specification of Software Quality Attributes," RADC-TR-85-37, Rome Air Development Center, Feb., 1985

SECTION VI

STUDY of the RELATIONSHIP BETWEEN METRIC DATA COLLECTION and 2167A

1. INTRODUCTION

The Rome Air Development Center has sponsored ongoing research into a software quality specification and measurement methodology, hereafter referred to as the "Methodology" The Methodology is documented in a three volume set of guidebooks:

— *Specification of Software Quality Attributes*, RADC-TR-85-37, Vols. I–III. [1]

Volume III presents a software quality evaluation methodology for determining the achieved levels of the software quality factors in intermediate and final software products. The evaluation methodology is based primarily on metric data collected during each phase of the software development process. The data source is a series of metric questions which are contained on a set of seven metric worksheets. Typically, the terminology used in the worksheets is consistent with DOD-STD-2167 [2], which was the standard in force at the time the guidebooks were written.

The seven worksheets are used to evaluate development products at different development phases and at different levels of detail.

- Worksheet 0, System level, system/software requirements analysis
- Worksheet 1, CSCI level, software requirements analysis
- Worksheet 2, CSCI level, preliminary design
- Worksheet 3A, CSCI level, detailed design
- Worksheet 3B, Unit level, detailed design
- Worksheet 4A, CSCI level, code and unit testing
- Worksheet 4B, Unit level, code and unit testing

Two sets of worksheets, 3A/3B and 4A/4B, span both the CSCI and Unit levels. The information required to answer many of the questions on the 3A or 4A CSCI level worksheets is collected on corresponding questions located on the 3B or 4B Unit level worksheets. Most metric questions are applicable to more than one development phase and so are found on more than one worksheet. The complete set of worksheets and the metric questions are found in Appendix A of Volume III.

The objective of this study was to develop a cross reference listing between each of the metric questions and the DOD-STD-2167A Data Item Descriptions (DIDs) [3] document and paragraph location where the information to answer the metric question was most likely to be found. We use the phrase *most likely to be found* literally. The content requirements and the specific language of the DOD-STD-2167A DIDs are written in such a way (at least in our view) so as to provide contractors with a reasonable latitude in complying with the standard. Unfortunately, this apparent degree of freedom made our task more difficult and introduced an undesirable but unavoidable element of judgement into the development of our cross reference guide. The next section presents the general rules which we developed in order to guide our work.

2. ANALYSIS GUIDELINES

1. For the metric questions contained in any specific worksheet, the only DID paragraphs which may be used as data sources for the metric questions as those paragraphs which are found in the documents directly related to the development phase covered by the specific worksheet.

Section 5 of DOD-STD-2167A, entitled Detailed Requirements, is organized into several sections, each section (X) corresponding to one of the DOD-STD-2167A software development phases. Section 5.X.4, Software Product Evaluations, lists the documents considered to be directly related to each development phase.

2. The Software Development Plan document may apply to all worksheets; however, its scope is limited to questions of standards or life cycle requirements which are not, in themselves, phase-unique.

3. Whenever possible, a single DID paragraph will be referenced as the data source to be used for answering metric questions. In cases where no single DID section contains adequate information, multiple paragraphs/documents may be referenced.

3. FINDINGS and RECOMMENDATIONS

3.1 Findings

In general, we found the process of identifying DOD-STD-2167A DID data sources for metric questions to be both difficult and frustrating. Clearly, a contributing factor was the quality of many of the questions. We did not attempt to formally categorize problems related to the wording and meaning of the questions themselves; others have already done so [4]. Also, we found many instances where worksheets contained inappropriate questions. For example, worksheet 0 questions which require information found in the DOD-STD-2167A System Design Document. (A few of these instances can be traced to the restructuring of the DIDs which accompanied the issuance of DOD-STD-2167A; recall that the methodology guidebooks were prepared at a time when the prior DOD-STD-2167 was in force). We have catalogued these and the other types of problems encountered; see the remarks labeled comments in section 4.1 below.

The other major factor which contributed to the task's difficulty was the structure of the DOD-STD-2167A DIDs and lack of supplemental interpretative and guidance material. As for the later, we have reviewed a nearly completed draft of an implementation guide for DOD-STD-2167A [5]. Unfortunately, this handbook provided almost no interpretation of DID requirements.

We also detected several basic incompatibilities between the Worksheets and DOD-STD-2167A. First, Worksheet 0 contains many questions related to system-wide technical requirements. For example, question EC.1(1): Have performance requirements and limitations for system communication efficiency been specified for each system function? While we can imagine this type of information being recorded in the DOD-STD-2167A System/Segment Specification, in either subparagraph 10.1.5.2.1.1.1.1 or 10.1.5.2.3.1 (see page 6 of the SSS DID), we are not entirely comfortable in the use of these paragraphs for the purpose of recording system-wide technical requirements. We did consider the use of SSS paragraph 10.1.5.2.5.4, Additional Quality Factors, as an alternative to the two subparagraphs mentioned above. However, with the issuance of DOD-STD-2168 [6], and the subsequent movement of software quality concerns out of DOD-STD-2167A and into DOD-STD-2168, we did not believe that the Additional Quality Factors paragraph would receive attention (but see recommendation 4 in section 3,2 below).

Second, many metric questions are concerned with user capabilities, especially in the area of user control over some system resource. The key DID documents, the SSS, SSDD, SRS, SDD (see section 4.1 below for the meanings of these acronyms), do not contain explicit requirements for reporting user control capabilities.

Third, while many metric element questions were related to detailed and significant technical matters, we often discovered that the (most) appropriate DID did not explicitly require that these

technical matters be recorded. For example, the SRS has no explicit requirement for reporting the processing and/or memory requirements related to separate capabilities.

3.2 *Recommendations*

1. Develop a metric questions guidebook. The objective of this guidebook is to facilitate the answering of the metrics questions. The guidebook should contain, for each metric question, discussions of the question's intent and narrow definitions of terms, what data is required, the DID or other reference(s) locations of the required data, and a procedure for using the data to answer the question. Short examples should be included for additional clarification.

2. Develop a DOD-STD-2167 guidebook for users of the Methodology. The objective of this guidebook is to facilitate understanding of DOD-STD-2167A from the perspective of an individual responsible for determining answers to metric questions. This guidebook should be written as a companion to the metrics questions guidebook.

3. Require future development of metric questions to follow a standard process. The objective of this standard process would be an orderly and evolutionary development of the metric questions set. As part of the process, each question should be reviewed for consistency with the most recent version of the *Military Standard Defense System Software Development* standard. This review should include both terminology consistency and a finding that the data required to answer the question is readily available.

4. Submit modification requests for DOD-STD-2167A. Since a major objective of DOD-STD-2167A is to increase the quality of delivered software, reviews of DOD-STD-2167A by individuals working to evolve the Methodology should result in suggestions to improve this standard. For example, we recommend that SSS paragraph 10.1.5.2.5.4, Additional Quality Factors, be renamed to Additional System-Wide Requirements. We believe that the suggested paragraph title more accurately reflects the intent of the SSS document.

4. INTRODUCTION to the CROSS REFERENCE GUIDE

4.1 Organization of the Cross Reference Guide

The material is presented in worksheet order. Each successive worksheet begins a new page. If one DID reference is sufficient to answer the metric question, then the entire entry is shown on one line. If two DID references are required, the second reference is shown immediately below the first. Strongly related questions are shown grouped together (no blank lines between questions).

4.2 Explanation of Column Headings

Metric Element. Each metric is defined by one or more metric elements (questions). The questions are shown in worksheet order. The question identifiers (acronyms) are used consistently throughout Guidebook Volume III. The two alphabetic characters identify a software criterion. (The software quality factors are defined in terms of these software criterion). The following integer identifies a metric attribute of the criterion. The following parentheses specify the metric elements which define the metric criteria.

DID. DOD-STD-2167A Data Item Description Acronyms (alphabetical order)

CRISD	Computer Resources Integrated Support Document
CSOM	Computer System Operator's Manual
FSM	Firmware Support Manual
IDD	Interface Design Document
IRS	Interface Requirements Specification
SDD	Software Design Document
SDP	Software Development Plan
SPM	Software Programmer's Manual
SPS	Software Product Specification
SRS	Software Requirements Specification
SSDD	System/Segment Design Document
SSS	System/Segment Specification
STD	Software Test Description
STP	Software Test Plan
STR	Software Test Report
SUM	Software User's Manual
VDD	Version Description Document

DID Paragraph. Each DID is organized into a set of numbered paragraphs. Each paragraph sets forth information reporting requirements. The paragraph number shown in the cross-reference listings for a given DID identifies the reporting requirements which most likely contain the data or information required to answer the corresponding metric element (question).

Document Reference. Each DID paragraph specifies the paragraph numbering which the reporting contractor shall use in order to identify the location of required information.
Comments. This column briefly states problem items mainly related to the wording of metric questions, the correspondence between software development phase and the level of specific worksheet questions, and certain other problematic items. Following is an explanation of the most frequent comment types.

- Inappropriate level (DID). Indicates that the metric question should not be asked on this worksheet, since the information required to answer the question is found in a DID not available at the development phase represented by the worksheet (see item 1 in section 2 above) Shows, in parentheses, the DID which most likely contains the information required to answer the question.

- Vague; term (X). Indicates that the question is vague due to the presence of term, or word X.

- Subjective. Indicates that the question is vulnerable to interpretation or judgmental influence.

- Indeterminate. Indicates that additional information from sources other than the DIDs is required to answer the question.

- Weak. Indicates that the assignment of the DID reference to the question was the best choice available, but that the assignment was made with low confidence

- ? Indicates a confusing, ambiguous or unresolvable question.

- Note: *References 3B* or *References 4B* are not indicators of problematic questions. These indicate that the data sources for the CSCI level 3A or 4A questions are the corresponding Unit level 3B or 4B questions.

5. REFERENCES

[1] T.P. Bowen, G. B. Wigle, and J. T. Tsai "Specification of Software Quality Attributes," RADC-TR-85-37, Rome Air Development Center, Feb., 1985

[2] DOD-STD-2167,*Military Standard Defense System Software Development,* June, 1985

[3] DOD-STD-2167A, *Military Standard Defense System Software Development,* Feb.,1988

[4] Patricia Pierce, Richard Hartley, and Suellen Worrells, "Software Quality Measurement Demonstration Project II," Final Technical Report, RADC-TR-87-164, Oct., 1987

[5] DOD-HDBK-287, *Military Handbook Defense System Software Development (Working Copy),* October 30, 1987

[6] DOD-STD-2168, *Defense Systems Software Quality Program,* April, 1988.

Cross Reference Guide

Between

DOD-STD-2167A

and

Metric Element Questions

Appendix A

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
AC.1(1)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	
AC.1(2)	SSDD	10.1.9	4	
AC.1(3)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	May be CSCI level (ie. SRS 10.1.5.4)
AC.1(4)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	May be CSCI level (ie. SRS 10.1.5.4)
AC.1(5)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	
AC.1(6)	SDP	10.2.6.3	4.3	Seems like s CSU level question
AM.1(1)				Inappropriate level (SRS)
AM.1(2)				Inappropriate level (SDD)
AM.1(3)	SDP	10.2.6.2.3	4.2.3	
AM.1(4)				Inappropriate level (SDD)
AM.2(1)	SRS SSS	10.1.5.4 10.1.5.2.3.1	3.4 3.2.3.X	Vague; term (external)
AM.3(1)	SSS SSS	10.1.5.2.5.1 10.1.3.2.5.1		(weak)
AM.4(1)	SSS SSS	10.1.5.2.5.1 10.1.3.2.5.1		(weak)
AM.5(1)	SSS SSS	10.1.5.2.5.1 10.1.3.2.5.1		(weak)

WS(0)-1

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
AM.6(1)	SSS	10.1.5.2.3.1	3.2.3.X	
AM.7(1)	SSS	10.1.5.2.3.1	3.2.3.X	
AM.7(2)	SSS	10.1.5.2.3.1	3.2.3.X	
AM.7(3)	SSS	10.1.5.2.3.1	3.2.3.X	
AP.1(1)	SDP	10.2.6.2.3	4.2.3	Vague level, seems system-wide
AP.2(2)	SDP	10.2.6.2.4		
AP.2(4)	SDP	10.2.6.2.4		
AP.3(1)	SDP	10.2.6.2.3	4.2.3	Vague level, seems system-wide
AP.4(1)	SSDD	10.1.7.1	5.X	
AP.5(1)	SSS	10.1.5.2.5.4	3.2.5.4	AQF (reusabilty)
AT.1(2)	SSS SSS	10.1.5.3.11 10.1.5.2.8	3.3.11 3.2.8	
AT.1(3)	SSS SSS	10.1.5.3.11 10.1.5.2.8	3.3.11 3.2.8	
AT.2(3)	SSS SSS	10.1.5.3.11 10.1.5.2.8	3.3.11 3.2.8	

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
AT.3(1)	SSS	10.1.5.3.11	3.3.11	
	SSS	10.1.5.2.8	3.2.8	
AT.3(2)	SSS	10.1.5.3.11	3.3.11	
	SSS	10.1.5.2.8	3.2.8	
AT.4(1)	SSS	10.1.5.2.8	3.2.8	(weak)
AT.4(2)	SSS	10.1.4	2	
AT.4(3)	SSS	10.1.4	2	
AU.1(1)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	Subjective
	SSDD	10.1.6.2.1		
	SSDD	10.1.6.4		
AU.2(1)	SSDD	10.1.7.1	5.X	Specific hardware questions (delete)
AU.2(2)	SRS	10.1.5.2.1	3.2.X	Vague term (executive), scope?
CL.1(1)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	
CL.1(2)	SDP	10.2.6.2.3	4.2.3	
	SSS	10.1.5.2.3		
CL.1(3)	SDP	10.2.6.2.3	4.2.3	
	SSS	10.1.5.2.3		
CL.1(4)	SDP	10.2.6.2.3	4.2.3	
	SSS	10.1.5.2.3		

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
CL.1(5)	SDP SSS	10.2.6.2.3 10.1.5.2.3	4.2.3	
CL.1(6)	SDP SSS	10.2.6.2.3 10.1.5.2.3	4.2.3	
CL.1(7)	SSDD SSDD	10.1.5.4 10.1.6.2.1		
CL.1(8)	SSDD SSDD	10.1.5.4 10.1.6.2.1		
CL.1(9)				Inappropriate level (IDD)
CL.1(10)	SSS	10.1.5.2.3.1	3.2.3.X	May need to reference IDD
CL.1(11)	SSS	10.1.5.2.3.1	3.2.3.X	Subjective; may need to reference IDD; term (data freshness)
CL.1(12)	SSS	10.1.5.2.3.1	3.2.3.X	
CL.1(13)	SSDD	10.1.5	3	Note: 2167A Operational Concepts is abbreviated in scope compared to 2167 Operational Concepts Document
CL.1(14)				Inappropriate level (CSOM)
CL.2(1)	SDP	10.2.6.2.3	4.2.3	
CL.2(2)				Inappropriate level (SDD); term (data translation)
CL.2(3)	SSS	10.1.5.2.3.1	3.2.3.X	
CL.2(4)				Inappropriate level (IDD)

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
CL.2(5)	SSS	10.1.5.2.3.1	3.2.3.X	
CL.2(6)				Inappropriate level (IDD)
CL.2(7)				Inappropriate level (IDD)
CL.2(8)				Inappropriate level (IDD)
CL.3(1)				Not mentioned in SDP (10.2.6.2)
CP.1(1)	SRS	10.1.5.2.1	3.2.X	Subjective; question reflects 2167 SRS 10.2.5.4.1 (.1,.2,.3)
CP.1(2)				Inappropriate level for preliminary SRS
CP.1(3)				Same comment as CP.1(2); part (b) requires SDD
CP.1(5)				Vague; term (referenced); may be SSDD, seems inappropriate level (SDD)
CP.1(6)	SSDD	10.1.6.1.1-3.1		
CP.1(7)				Vague; term (referenced); inappropriate level (SDD).
CP.1(8)				Inappropriate level (SDD)
CP.1(11)				Not a deliverable
CS.1(1)	SDP	10.2.6.2.3	4.2.3	(Weak)
CS.1(2)	SDP	10.2.6.2.3	4.2.3	(Weak)
CS.1(3)	SDP	10.2.6.2.3	4.2.3	(Weak)
CS.1(4)	SDP	10.2.6.2.3	4.2.3	(Weak)
CS.1(5)				?; term (references); level uncertain
CS.2(1)	SDP	10.2.6.2.3	4.2.3	
CS.2(2)	SDP	10.2.6.2.4	4.2.3	

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
CS.2(3)	SDP	10.2.6.2.3	4.2.3	
CS.2(4)				Seems to be a post deployment matter
CS.2(5)				Seems to be a post deployment matter
CS.2(6)				?; term (references)
DI.1(1)	SSDD	10.1.6.2.1	4.2.X	DID has no stated requirement for graphic presentation
DI.1(2)	SSDD	10.1.5.5	3.5	

Note: The following group of DI questions are system-wide technical requirements for availability/survivability/operability. They can be most naturally documented in SSS 10.1.5.2.5.4, Additional Quality Factors, (doc ref 3.2.5.4), with a suggested DID paragraph name change to Additional System-Wide Requirements.

DI.1(3)				
DI.1(4)				
DI.1(5)				
DI.1(6)				
DI.1(7)				
DI.1(8)				
D1.1(9)				
DO.1(1)	SDP	10.2.5.9	3.9	
DO.2(1)				Vague question; subjective; level uncertain. Better question: "Does documentation conform to documentation standards?"
DO.2(2)	SSDD	10.1.5.5	3.5	Does not meet requirements segment of question
DO.2(3)	All	10.1.2		

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
DO.2(4)	SSS SSDD	10.1.3.3 10.1.3.3	1.3 1.3	Level not clear
DO.2(5)	SRS	10.1.5.2.1	3.2.X	Subjective; term (completely characterize)
DO.2(6)				Inappropriate level

Note: The following group of EC, EP, and ES questions are system-wide technical requirements for efficiency. They can be most naturally documented in SSS 10.1.5.2.5.4, Additional Quality Factors, (doc ref 3.2.5.4), with a suggested DID paragraph name change to Additional System-Wide Requirements. Alternatively, this EC/EP/ES group can be documented as shown below

EC.1(1)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	
EP.1(1)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	
EP.1(3)	SDP SSDD	10.2.6.1.3.1 10.1.7.1	4.1.3.1 5.X	
EP.1(5)	SRS	10.1.5.9	3.9	(Weak) question pertains to system memory management
EP.2(1)				Inappropriate level (SDD)
EP.2(2)				Vague, not testable
EP.2(3)	SDP	10.2.6.2.4	4.2.4	
ES.1(1)				Inappropriate level (SRS); vague; not testable
ES.1(2)	SSDD	10.1.7.1	5.X	
ES.1(5)	SSDD	10.1.7.1	5.X	

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
ES.1(7)	SDP SSDD	10.2.6.1.3.1 10.1.7.1	4.1.3.1 5.X	
ES.1(8)				Vague; term (avoid)
FO.1(1)a	SSS	10.1.5.2.2	3.2.2	
FO.1(1)b				Indeterminate
FO.1(2)				Indeterminate
FO.1(3)				Indeterminate
FO.1(4)				Indeterminate
FS.2(1)	SSS	10.1.5.2.5.4	3.2.5.4	Additional Quality Factor (Reusability)
FS.2(2)				Indeterminate; need explict criteria to answer part (b)
FS.2(3)				Inappropriate level(see SRS); term (inputs)
FS.2(4)				Inappropriate level(see SRS)
FS.2(5)				Inappropriate level(see SRS); term (outputs)
FS.2(6)				Indeterminate; need explict criteria to answer part (b)
FS.3(1)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	(Weak)
FS.3(2)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	(Weak)
FS.3(3)				Subjective; term (typically); indeterminate
ID.1(2)				Inappropriate level (Statement of Work)
ID.1(3)	SDP	10.2.6.2.4	4.2.4	
ID.2(1)				Indeterminate

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
MO.1(1)	SDP	10.2.6.2.1	4.2.1	Not required in 2167A; specific methodology
MO.1(2)				Inappropriate level (SDD)
MO.2(1)	SDP	10.2.6.2.3	4.2.3	
MO.2(4)	SDP	10.2.6.2.3	4.2.3	
OP.1(1)	SSS	10.1.5.2.2	3.2.2	
	SSDD	10.1.5.5		
OP.1(2)				Inappropriate level (SUM)
OP.1(3)				Inappropriate level (SUM)
OP.1(4)				This use of term "capability" not explicitly found
OP.1(5)				Inappropriate level (CSOM); indeterminate
OP.1(6)	SSS	10.1.5.3.7	3.11	
OP.1(8)	SSS	10.1.5.3.7	3.11	
OP.1(10)	SSS	10.1.5.3.7	3.11	
OP.1(11)	SSS	10.1.5.2.5.4		
OP.1(12)	SSS	10.1.5.2.5.4		
OP.1(13)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	Level lower than system; may be redundant with Interoperability section
OP.1(14)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	Level lower than system; may be redundant with Interoperability section
OP.1(15)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	Level lower than system; may be redundant with Interoperability section
OP.1(16)				Not found; Duplicates VR.1(1)
OP.2(1)				Inappropriate level

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
OP.2(4)	SSS	10.1.5.3.7	3.11	
OP.2(5)				Vague; term (logical end of input)
OP.2(6)	SSS	10.1.5.3.7	3.11	
OP.3(1)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	(Weak)
OP.3(2)	SSS	10.1.5.3.7	3.11	
OP.3(3)	SSS	10.1.5.3.7	3.11	
OP.3(6)	SSS	10.1.5.3.7	3.11	
OP.3(7)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	(Weak) Similar to OP.3(1)

Note: The following group of RE questions are system-wide technical requirements for availability/survivability. They can be most naturally documented in SSS 10.1.5.2.5.4, Additional Quality Factors, (doc ref 3.2.5.4), with a suggested DID paragraph name change to Additional System-Wide Requirements. Alternatively, this RE group can be documented as shown below

RE.1(1)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	
RE.1(2)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	
RE.1(3)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	
RE.1(4)	SSS	10.1.5.2.1.1.1.1	3.2.1.X.Y.Z	
SD.2(1)	SDP	10.2.6.2.4	4.2.4	
SD.2(2)	SDP	10.2.6.2.4	4.2.4	
SD.3(5)	SDP	10.2.6.2.3	4.2.3	
SI.1(1)	SSS	10.1.5.2.2	3.2.2	
SI.1(8)	Mandated by 2167A, Appendix B			Same as SI.4(13)
SI.1(9)	SDP	10.2.6.2.4	4.2.4	

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
SI.2(1)	SDP	10.2.6.1.3.1	4.1.3.1	
SI.4(13)	SDP	10.2.6.2.4	4.2.4	Same as SI.4(8)
SS.1(1)	SSS	10.1.5.2.5.4		
SS.1(2)	SSS	10.1.5.2.5.4		
SS.1(3)	SDP	10.2.6.2.3	4.2.3	
SS.1(4)	SSS	10.1.5.2.3	3.2.3	
SS.2(1)	SSS	10.1.5.2.5.4		
SS.2(2)	SSS	10.1.5.2.5.4		
ST.3(1)	SDP	10.2.6.2.3	4.2.3	
ST.3(2)				Inappropriate level (SDD)

Note: The following group of SY questions are system-wide technical requirements for availability/survivability/interoperability. They can be most naturally documented in SSS 10.1.5.2.5.4, Additional Quality Factors, (ref 3.2.5.4), with a suggested DID paragraph name change to Additional System-Wide Requirements. Alternatively, this SY group can be documented as shown below

SY.1(1)	SSS	10.1.5.2.3.1	3.2.3.1
SY.1(2)	SSS	10.1.5.2.3.1	3.2.3.1
SY.1(3)	SSS	10.1.5.2.3.1	3.2.3.1
SY.1(4)	SSS	10.1.5.2.3.1	3.2.3.1
SY.2(1)	SRS	10.1.5.4	3.4
SY.2(2)	SRS	10.1.5.4	3.4
SY.2(3)	SRS	10.1.5.4	3.4

Worksheet 0 - System Level
System Requirements Analysis/Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
SY.3(1)	SSS	10.1.5.2.3.1	3.2.3.X	Broad interpretation of term (external interface requirement)
SY.3(2)	SSDD	10.1.7.1	5.X	
SY.3(3)	SSS	10.1.5.2.3.1	3.2.3.X	Broad interpretation of term (external interface requirement)
SY.3(4)	SSDD	10.1.7.1	5.X	
SY.3(5)	SSS	10.1.5.2.3.1	3.2.3.X	Broad interpretation of term (external interface requirement)
SY.3(6)	SSDD	10.1.7.1	5.X	
SY.4(1)	SDP	10.2.6.1.3.1	4.1.3.1	
SY.4(2)	SDP	10.2.6.1.3.1	4.1.3.1	
SY.4(3)	CRISD	10.1.5.1.1	3.11	
SY.5(1)	SSS	10.1.4	2	Subjective
TN.1(1)	SSS	10.1.5.6.2	3.6.2	
TN.1(2)	SSS	10.1.5.6.2	3.6.2	
TN.1(3)	SSS	10.1.5.6.2	3.6.2	(Weak)
TN.1(4)	SSS	10.1.5.6.2	3.6.2	(Weak)
VR.1(1)				Same as OP.1(16)

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
AC.1(3)	SRS	10.1.5.4	3.4	
AC.1(4)	SRS	10.1.5.4	3.4	
AC.1(5)	SRS	10.1.5.4	3.4	
AC.1(6)	SDP	10.2.6.3	4.3	
AM.1(1)	SRS	10.1.5.2	3.2	
AM.1(2)				Indeterminate
AM.1(3)	SDP	10.2.6.2.3	4.2.3	
AM.1(4)	SRS	10.1.5.2.1	3.2.X	
AM.2(1)	IRS	10.1.5.2.2	3.2.2	
AM.3(1)				Inappropriate level (SDD)
AM.3(2)				Inappropriate level (SDD)
AM.3(3)				Inappropriate level (SDD)
AM.3(4)				Inappropriate level (SDD)
AM.4(1)	SRS	10.1.5.2.1	3.1	
AM.5(1)	SRS	10.1.5.1	3.1	
AM.6(1)	SRS	10.1.5.1	3.1	

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
AM.7(1)	SRS	10.1.5.1	3.1	
AM.7(2)	SRS	10.1.5.1	3.1	
AM.7(3)	SRS	10.1.5.1	3.1	
AP.1(1)	SRS	10.1.5.2.1	3.2.X	(Weak)
AP.2(2)	SDP	10.2.6.2.4	4.2.4	
AP.2(4)	SDP	10.2.6.2.4	4.2.4	
AP.3(1)	SRS	10.1.5.2.1	3.2.X	
AP.4(1)	SRS	10.1.5.9	3.9	
AP.5(1)a	SRS	10.1.5.2.1	3.2.X	
AP.5(1)b				Indeterminate
AT.1(2)	SRS	10.1.5.6	3.6	SRS 10.1.5.6 does not explicitly mention reserve capacity requirements. Documentation of CSCI reserves may be required here in order to comply with 2167A general requirement 4.2.10
AT.1(3)	SRS	10.1.5.6	3.6	
AT.2(3)	SRS	10.1.5.6	3.6	
AT.3(1)	SRS	10.1.5.6	3.6	
AT.3(2)	SRS	10.1.5.6	3.6	
AT.4(1)	SRS	10.1.5.1	3.2.X	Assumes question is about software compatibilities
AT.4(2)	SRS	10.1.4	2	
AT.4(3)	SRS	10.1.4	2	
AU.1(1)				Inappropriate level (SDD)
AU.2(1)				Inappropriate level (SSDD); hardware requirement
AU.2(2)	SRS	10.1.5.2.1	3.2.X	

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
CL.1(1)				Inappropriate level (SSDD)
The following group assumes that "design standard" does not apply to process methodologies, but to actual products (see terminology, 2167A, Appendix D, 10.2.5). Regardless, these questions seem more pertinent to level 0 (where they were already asked). Do these questions mean to ask: "Do the IRS/SRS internal interface req'ts reflect the previously defined network standards?"				
CL.1(2)	SDP	10.2.6.2.3	4.2.3	
CL.1(3)	SDP	10.2.6.2.3	4.2.3	
CL.1(4)	SDP	10.2.6.2.3	4.2.3	
CL.1(5)	SDP	10.2.6.2.3	4.2.3	
CL.1(6)	SDP	10.2.6.2.3	4.2.3	
CL.1(7)	SRS	10.1.5.2.1	3.2.X	
CL.1(8)	SRS	10.1.5.2.1	3.2.X	
CL.1(9)				Inappropriate level (IDD)
CL.1(10)				Subjective; may need to reference IDD
CL.1(11)				Subjective; may need to reference IDD; term (data freshness)
CL.1(12)	SRS	10.1.5.1	3.1	
CL.1(13)				Inappropriate level (SSDD); Note: 2167A Operational Concepts is abbreviated in scope compared to 2167 Operational Concepts Document.
CL.1(14)				Inappropriate level (CSOM)

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
CL.2(1)	SDP	10.2.6.2.3	4.2.3	Two part question : (1) data representation, (2) data translation.
CL.2(2)				Inappropriate level (SDD)
CL.2(3)	SRS	10.1.5.1	3.1	(Will reference IRS documents)
CL.2(4)				Inappropriate level (IDD)
CL.2(5)	SRS	10.1.5.1	3.1	(Will reference IRS documents)
CL.2(6)				Inappropriate level (IDD)
CL.2(7)				Inappropriate level (IDD)
CL.2(8)				Inappropriate level (IDD)
CL.3(1)				Not mentioned in SDP (10.2.6.2)
CP.1(1)	SRS	10.1.5.2.1	3.2.X	Subjective; question reflects 2167 SRS 10.2.5.4.1
CP.1(2)	SRS	10.1.5.4	3.4	"data references" same as data types mentioned in 10.1.5.2.1?
CP.1(3)	SRS	10.1.5.4	3.4	Similar to CP.1(2)
CP.1(5)				Vague; term (referenced); inappropriate level (SDD). Seems to ask: "Have all capabilities in 10.1.5.2.1 been mentioned elsewhere in this (or another) document?" If so, this is answered during a review meeting, not by a metric.
CP.1(6)	SRS	10.1.5.2.1	3.2.X	
	SRS	10.1.5.12	3.12	
CP.1(7)				Same as CP.1(5);Vague; term (referenced); inappropriate level (SDD)
CP.1(8)				Inappropriate level (SDD)

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
CP.1(11)				Not a deliverable
CS.1(1)	SDP	10.2.6.2.3	4.2.3	Same argument as for CL.1(2)-(6)
CS.1(2)	SDP	10.2.6.2.3	4.2.3	
CS.1(3)	SDP	10.2.6.2.3	4.2.3	
CS.1(4)	SDP	10.2.6.2.3	4.2.3	
CS.1(5)				?; term (references); just a text search?
CS.2(1)	SDP	10.2.6.2.3	4.2.3	
CS.2(2)	SDP	10.2.6.2.3	4.2.3	
CS.2(3)	SDP	10.2.6.2.3	4.2.3	
CS.2(4)				Seems to be a post deployment matter
CS.2(5)				Seems to be a post deployment matter
CS.2(6)				?; term (references); just a text search?
DI.1(1)	SRS	10.1.5.2	3.2.X	
DI.1(2)	SRS	10.1.5.3	3.3	

Note: The following group of DI questions are system-wide technical requirements for availability/survivability. They can be most naturally documented in SRS 10.1.5.10, Software Quality Factors, (doc ref 3.10), with a suggested DID paragraph name change to Additional Software Requirements. Alternatively, the DI group can be documented in SRS 10.1.5.2.1 (doc ref 3.2.X)

DI.1(3)
DI.1(4)
DI.1(6)
DI.1(7)
DI.1(8)

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
DI.1(9)				
DO.1(1)	SDP	10.2.5.9	3.9	
DO.2(1)				Subjective
DO.2(2)	SRS	10.1.5.3		Does not meet design segment of question
DO.2(3)	All	10.1.2		
DO.2(4)	SRS IRS	10.1.3.3 10.1.3.3	1.3 1.3	Does not answer design segment of the question
DO.2(5)	SRS	10.1.5.2.1	3.2.X	Subjective; term (completely characterize)
DO.2(6)				Inappropriate level

Note: The following group of EC, EP, and ES questions are system-wide technical requirements for efficiency. They can be most naturally documented in SRS 10.1.5.10, Software Quality Factors, (doc ref 3.10), with a suggested DID paragraph name change to Additional Software Requirements. Alternatively, this EC/EP/ES group can be documented as shown below

EC.1(1)	SRS	10.1.5.2.1	3.2.X	
EP.1(1)	SRS	10.1.5.2.1	3.2.X	
EP.1(3)				Inappropriate level (SSDD); it's in SDD, so must be in SRS.
EP.1(5)	SRS	10.1.5.9	3.9	
EP.2(1)	SRS	10.1.5.6	3.6	(Weak) This offers no breakdown of capabilities
EP.2(2)				Vague; not testable (software quality factors?)

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
EP.2(3)	SDP	10.2.6.1.3.1		
ES.1(1)	SRS	10.1.5.6	3.6	Vague; not testable
ES.1(2)				Inappropriate level (SSDD)
ES.1(5)				Inappropriate level (SSDD)
ES.1(7)				Inappropriate level (SSDD)
ES.1(8)	SRS	10.1.5.9		
FO.1(1)a	SRS	10.1.5.2	3.2.X	Consistent with FS.2(2)-(5)a?
FO.1(1)b				Indeterminate
FO.1(2)				Indeterminate
FO.1(3)				Indeterminate
FO.1(4)				Indeterminate
FS.2(1)	SRS	10.1.5.10	3.10	Reusability
FS.2(2)a	SRS	10.1.5.2.1	3.2.X	
FS.2(2)b				Indeterminate
FS.2(3)	SRS	10.1.5.2.1	3.2.X	Vague; term (limitations);asks two questions.
		10.1.5.4	3.4	
FS.2(4)	SRS	10.1.5.2.1	3.2.X	See FS.2(3)
		10.1.5.4	3.4	
FS.2(5)	SRS	10.1.5.2.1	3.2.X	See FS.2(3)

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
		10.1.5.4	3.4	
FS.2(6)				Indeterminate; need explicit criteria to answer part (b)
FS.3(1)	SRS	10.1.5.11		
FS.3(2)				Scope: "modificatic of resources allocated"?
FS.3(3)				Subjective; term (typically); Indeterminate
ID.1(2)	SDP	10.2.6.2.4	4.2.4	
	SDP	10.2.6.1.3.1		
ID.1(3)	SDP	10.2.6.2.4	4.2.4	
	SDP	10.2.6.1.3.1		
ID.2(1)				Indeterminate
MO.1(1)	SDP	10.2.6.2.3	4.2.3	
MO.1(2)				Inappropriate level (SDD)
MO.2(1)	SDP	10.2.6.2.3	4.2.3	Same argument as CL.1(2)-(6)
MO.2(2)				Inappropriate level (SDD)
MO.2(3)				Inappropriate level (SDD)
MO.2(4)	SDP	10.2.6.2.3	4.2.3	Same argument as CL.1(2)-(6)
MO.2(5)	SRS	10.1.5.2.1	3.2.X	
OP.1(1)	SRS	10.1.5.2.1	3.2.X	

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
OP.1(2)				Inappropriate level (SUM)
OP.1(3)				Inappropriate level (SUM)
OP.1(4)	SRS	10.1.5.2.1	3.2.X	(Weak); "user capability"=CSCI capability/function?
OP.1(5)				Inappropriate level (CSOM); indeterminate
OP.1(6)	SRS	10.1.5.11	3.11	
OP.1(8)	SRS	10.1.5.11	3.11	
OP.1(10)	SRS	10.1.5.11	3.11	
OP.1(11)	SRS	10.1.5.8	3.8	
OP.1(12)	SRS	10.1.5.8	3.8	
OP.1(13)	SRS	10.1.5.2.1	3.2.X	(Weak); "user capability"=CSCI capability/function?
OP.1(14)	SRS	10.1.5.2.1	3.2.X	(Weak); "user capability"=CSCI capability/function?
OP.1(15)	SRS	10.1.5.2.1	3.2.X	(Weak); "user capability"=CSCI capability/function?
OP.1(16)	SRS	10.1.5.11	3.11	Same as VR.1(1)
OP.2(1)				Inappropriate level (SDD)
OP.2(2)				Inappropriate level (SDD)
OP.2(3)				Inappropriate level (SDD)
OP.2(4)	SRS	10.1.5.2.1	3.2.X	(Weak); "user capability"=CSCI capability/function?
OP.2(5)				Vague; term (logical end of input)
OP.2(6)	SRS	10.1.5.11		
OP.3(1)	SRS	10.1.5.11		

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
OP.3(2)	SRS	10.1.5.11	3.11	
OP.3(3)	SRS	10.1.5.11	3.11	
OP.3(4)				Inappropriate level (SDD)
OP.3(5)	SRS	10.1.5.11	3.11	Subjective
OP.3(6)	SRS	10.1.5.11	3.11	
OP.3(7)				Similar to OP.3(1)

Note: The following group of RE questions are system-wide technical requirements for availability/survivability. They can be most naturally documented in SRS 10.1.5.10, Software Quality Factors, (doc ref 3.10), with a suggested DID paragraph name change to Additional Software Requirements. Alternatively, this RE group can be documented as shown below

RE.1(1)				
RE.1(2)				
RE.1(3)				
RE.1(4)				
SD.2(1)	SDP	10.2.6.2.4	4.2.4	
SD.2(2)	SDP	10.2.6.2.4	4.2.4	
SD.3(5)	SDP	10.2.6.2.3	4.2.3	
SI.1(1)	SRS	10.1.5.2	3.2	
SI.1(8)	Mandated by 2167A; Appendix B			Same as SI.4(13)
SI.1(9)	SDP	10.2.6.2.4	4.2.4	
SI.2(1)	SDP	10.2.6.1.3.1		

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
SI.4(13)				Same as SI.1(8)
SS.1(1)	SRS	10.1.5.8	3.8	
SS.1(2)	SRS	10.1.5.8	3.8	
SS.1(3)	SDP	10.2.6.2.3	4.2.3	
SS.1(4)	SRS	10.1.5.8	3.8	
SS.2(1)	SRS	10.1.5.8	3.8	
SS.2(2)	SRS	10.1.5.8	3.8	
ST.3(1)	SDP	10.2.6.2.3	4.2.3	
ST.3(2)				Inappropriate level (SDD)

Note: The following group of SY questions are system-wide technical requirements for availability/survivability. They can be most naturally documented in SRS 10.1.5.10, Software Quality Factors, (doc ref 3.10), with a suggested DID paragraph name change to Additional Software Requirements. Alternatively, this SY group can be documented as shown below

SY.1(1)	SRS	10.1.5.1	3.1	
SY.1(2)	SRS	10.1.5.1	3.1	
SY.1(3)	SRS	10.1.5.1	3.1	
SY.1(4)	SRS	10.1.5.1	3.1	
SY.2(1)	SRS	10.1.5.4	3.4	
SY.2(2)	SRS	10.1.5.4	3.4	
SY.2(3)	SRS	10.1.5.4	3.4	
SY.3(1)				Inappropriate level (SSDD)
SY.3(2)				Inappropriate level (SSDD)
SY.3(3)				Inappropriate level (SSDD)

Worksheet 1 - CSCI Level
Software Requirements Analysis

Metric Element	DID	DID Paragraph	Document Reference	Comments
SY.3(4)				Inappropriate level (SSDD)
SY.3(5)				Inappropriate level (SSDD)
SY.3(6)				Inappropriate level (SSDD)
SY.4(1)	SDP	10.2.6.1.3.1	4.1.3.1	
SY.4(2)	SDP	10.2.6.1.3.1	4.1.3.1	
SY.4(3)	CRISD	10.1.5.1.1	3.1.1	Wrong phase of life cycle
SY.5(1)	SRS	10.4.1	2	Subjective
TC.1(1)	SRS	10.1.5.12	3.12	
TN.1(1)	SRS	10.1.5.11	3.11	
TN.1(2)	SRS	10.1.5.11	3.11	
TN.1(3)	SRS	10.1.5.11	3.11	
TN.1(4)	SRS	10.1.5.11	3.11	
VR.1(1)				Same as OP.1(16)

Worksheet 2 - CSCI Level
Preliminary Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
AC.1(7)				Inappropriate level (SDD detailed design)
AM.3(1)	SDD	10.1.5.2.1	3.2.X	
AM.4(1)	SDD	10.1.5.2.1	3.2.X	
AM.5(1)	SDD	10.1.5.2.1	3.2.X	
AM.6(1)	IDD	10.1.5.2.4.1	3.X.4.Y	
AM.6(2)	IDD	10.1.5.2.4.1	3.X.4.Y	
AM.6(3)	IDD	10.1.5.2.4.1	3.X.4.Y	
AM.6(4)	IDD	10.1.5.2.4.1	3.X.4.Y	
AM.7(1)	SDD	10.1.5.2.1	3.2.X	May require IDD
AM.7(2)	SDD	10.1.5.2.1	3.2.X	May require IDD
AM.7(3)	SDD	10.1.5.2.1	3.2.X	May require IDD
AP.1(1)	SDD	10.1.5.2.1	3.2.X	May require IDD
AP.5(1)a				Inappropriate level (SDD detailed design)
AP.5(1)b				Subjective; Inappropriate level (SDD detailed design)
AP.5(2)a				Inappropriate level (SDD detailed design)
AP.5(2)b				Inappropriate level (STP)
AP.5(3)a				Inappropriate level (SDD detailed design)
AP.5(3)b				Inappropriate level (STR)
AT.1(2)a	SDD	10.1.5.1.3	3.1.3	
AT.1(2)b	SDD	10.1.5.2.1	3.2.1	Look at design for estimate
AT.1(3)a	SDD	10.1.5.1.3	3.1.3	
AT.1(3)b	SDD	10.1.5.2.1	3.2.1	Look at design for estimate

Worksheet 2 - CSCI Level
Preliminary Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
AT.2(3)a	SDD	10.1.5.1.3	3.1.3	
AT.2(3)b	SDD	10.1.5.2.1	3.2.1	Look at design for estimate
AT.3(1)a	SDD	10.1.5.1.3	3.1.3	
AT.3(1)b	SDD	10.1.5.2.1	3.2.1	Look at design for estimate
AT.3(2)a	SDD	10.1.5.1.3	3.1.3	
AT.3(2)b	SDD	10.1.5.2.1	3.2.1	Look at design for estimate
AT.4(1)				Inappropriate level; H/W
AU.1(1)	SDD	10.1.5.1.1	3.1.1	Subjective
AU.1(4)a	SDD	10.1.5.1.3	3.1.3	
AU.1(4)b	IDD	10.1.5.2.4.1	3.X.4.Y	Vague; term (hardware)
AU.2(2)	SDD	10.1.5.2.1	3.2.X	Assumes "in accordance with requirements"
CL.1(2)	IDD	10.1.5.2.4.1	3.X.4.Y	
CL.1(3)	IDD	10.1.5.2.4.1	3.X.4.Y	
CL.1(4)	IDD	10.1.5.2.4.1	3.X.4.Y	
CL.1(5)	IDD	10.1.5.2.4.1	3.X.4.Y	
CL.1(6)	IDD	10.1.5.2.4.1	3.X.4.Y	
CL.1(7)a	SDD	10.1.5.2.1	3.2.X	
CL.1(8)a	SDD	10.1.5.2.1	3.2.X	
CL.1(11)	IDD	10.1.5.2.1	3.X.1	
CL.1(13)				Inappropriate level (SDDD)
CL.1(14)				Inappropriate level (CSOM)

Worksheet 2 - CSCI Level
Preliminary Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
CL.2(1)	SDD	10.1.6.1.2.2	4.X.Y.2	Wrong phase (detailed)
	SDP	10.2.6.2.3	4.2.3	
CL.2(2)	SDD	10.1.5.1.1	3.1.1	May not be able to answer until detailed design is complete
CL.2(4)	IDD	10.1.5.2.4.1	3.X.4.Y	
CL.2(6)	IDD	10.1.5.2.4.1	3.X.4.Y	
CP.1(1)	SDD	10.1.6.1.2.2	4.X.Y.2	
CP.1(2)a	SDD	10.1.7	5	Wrong phase (detailed)
CP.1(2)b	SDD	10.1.7	5	Wrong phase (detailed)
CP.1(3)a	SDD	10.1.7	5	Wrong phase (detailed)
CP.1(3)b	SDD	10.1.7	5	Wrong phase (detailed)
CP.1(4)a	SDD	10.1.7	5	Wrong phase (detailed)
CP.1(4)a	SDD	10.1.7	5	Wrong phase (detailed)
CP.1(6)	SDD	10.1.5.2.1	3.2.X	
CP.1(9)	SDD	10.1.5.2.1	3.2.X	
CP.1(11)				Non-deliverable item
CS.1(1)	SDD	10.1.5.2.1	3.2.X	
	SDP	10.2.6.2.3	4.2.3	
CS.1(5)	SDD	10.1.5.2.1	3.2.X	

Worksheet 2 - CSCI Level
Preliminary Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
CS.2(1)	SDD	10.1.7	5	Wrong phase (detailed)
	SDP	10.2.6.2.4	4.2.4	Wrong phase (detailed)
CS.2(2)	SDD	10.1.7	5	Wrong phase (detailed)
	SDP	10.2.6.2.4	4.2.4	Wrong phase (detailed)
CS.2(3)	SDD	10.1.7	5	Wrong phase (detailed)
	SDP	10.2.6.2.4	4.2.4	Wrong phase (detailed)
CS.2(4)				Seems to be a post deployment matter
CS.2(5)				Seems to be a post deployment matter
CS.2(6)	SDD	10.1.7	5	Term (references);wrong phase (detailed)
DL.1(1)	SDD	10.1.5.1.1	3.1.1	
DL.1(2)	SDD	10.1.5.1.2	3.1.2	
DL.1(4)	SDD	10.1.5.2.1	3.2.X	(Weak) May be 10.1.5.1.2
DL.1(5)	SDD	10.1.5.2.1	3.2.X	(Weak) May be 10.1.5.1.2
DL.1(6)	SDD	10.1.5.2.1	3.2.X	(Weak) May be 10.1.5.1.2
DL.1(7)	SDD	10.1.5.2.1	3.2.X	(Weak) May be 10.1.5.1.2
DL.1(9)	SDD	10.1.5.2.1	3.2.X	
DO.1(1)	SDP	10.2.5.9	3.9	
DO.2(1)				Subjective
DO.2(2)	SDD	10.1.5.1.2	3.1.2	Graphic presentation not specified
DO.2(3)	All	10.1.2		

Worksheet 2 - CSCI Level
Preliminary Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
DO.2(4)	All	10.3.3	1.3	
DO.2(5)	SDD	10.1.5.2.1	3.2.X	Subjective; term (clearly)
DO.2(7)				Inappropriate level (SDD detailed design)
EP.1(5)				Inappropriate level (SDD detailed design)
EP.2(2)				Inappropriate level (SDD detailed design); subjective
EP.2(3)	SDP	10.2.6.2.4	4.2.4	
EP.2(6)				Subjective, depends on intended use
ES.1(2)	SDD	10.1.5.2.1	3.2.X	
ES.1(5)	SDD	10.1.5.2.1	3.2.X	
ES.1(8)	SDD SDD	10.5.1.1 10.1.5.2.1	3.1.1 3.2.X	
FS.2(2)				Subjective
FS.2(3)	SDD	10.1.7		(wrong phase) For global inputs/outputs; local inputs/outputs are described in SDD detailed design. Limitations and interpretations are not explicitly required in 10.1.7.
FS.2(4)	SDD	10.1.7		See FS.2(3)
FS.2(5)	SDD	10.1.7		See FS.2(3)

Worksheet 2 - CSCI Level
Preliminary Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
FS.3(1)	SDD	10.1.5.2.1	3.2.X	
FS.3(2)	SDD	10.1.5.2.1	3.2.X	
ID.1(3)	SDP	10.2.6.2.4	4.2.4	
ID.2(1)				*Indeterminant*
MO.1(2)	SDD	10.1.5.1.1	3.1.1	
MO.1(9)	SDD	10.1.5.1.1	3.1.1	
MO.2(2)a	SDD	10.1.5.1.1	3.1.1	
MO.2(2)b	SDD	10.1.5.2.1	3.2.X	
MO.2(3)a	SDD	10.1.5.1.1	3.1.1	
MO.2(3)b	SDD	10.1.5.1.2	3.1.2	
MO.2(5)	SDD	10.1.5.1.1 10.1.5.2.1	3.1.1 3.2.X	
NOTE: Most of the OP series questions are weakly defined.				
OP.1(1)				Inappropriate level (SUM)
OP.1(2)	SDD	10.1.5.2.1	3.2.X	
OP.1(3)	SDD	10.1.5.2.1	3.2.X	
OP.1(4)	SDD	10.1.5.2.1	3.2.X	
OP.1(7)				Inappropriate level (SUM)
OP.1(9)	SDD	10.1.5.2.1	3.2.X	

Worksheet 2 - CSCI Level
Preliminary Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
OP.1(10)				Inappropriate level (SUM); subjective
OP.1(11)	SDD	10.1.5.2.1	3.2.X	
OP.1(12)				Procedural question
OP.1(13)	SDD	10.1.5.2.1	3.2.X	
OP.1(14)	SDD	10.1.5.2.1	3.2.X	
OP.1(15)	SDD	10.1.5.2.1	3.2.X	
OP.1(16)	SDD	10.1.5.2.1	3.2.X	
OP.2(4)	SDD	10.1.5.2.1	3.2.X	
OP.2(5)				Vague; term (logical end of input)
OP.2(6)	SDD	10.1.5.2.1	3.2.X	
OP.3(1)	SDD	10.1.5.2.1	3.2.X	
OP.3(2)	SDD	10.1.5.2.1	3.2.X	Assumes "in accordance with requirements"
OP.3(3)	SDD	10.1.5.2.1	3.2.X	See above
OP.3(6)	SDD	10.1.5.2.1	3.2.X	See above
OP.3(7)	SDD	10.1.5.2.1	3.2.X	See above
OP.3(8)	SDP	10.2.6.2.3	4.2.3	
RE.1(1)	IDD	10.1.5.2.4.1		Assumes "in accordance with requirements"
RE.1(2)	SDD	10.1.5.2.1	3.2.X	See above
RE.1(3)	SDD	10.1.5.2.1	3.2.X	See above
RE.1(4)	SDD	10.1.5.2.1	3.2.X	See above

Worksheet 2 - CSCI Level
Preliminary Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
SI.1(1)	SDD	10.1.5.1.1	3.1.1	2167 requirement
SI.1(6)a	SDD	10.1.7	5	Wrong phase (detailed)
SI.1(7)a	SDD	10.1.7	5	Wrong phase (detailed)
SI.1(7)b	SDD	10.1.7	5	Wrong phase (detailed)
SI.1(9)	SDP	10.2.6.2.4	4.2.4	2167A mandatory; see 2167A Appendix B
SI.1(10)	SDD	10.1.5.1.1	3.1.1	Hardware not specified
SI.2(1)	SDP	10.2.6.1.3.1	4.1.3.1	
SI.4(13)				Same as SI.1(9)
NOTE: The following block of questions is weakly defined.				
SS.1(1)	SDD	10.1.5.2.1	3.2.X	
SS.1(2)	SDD	10.1.5.2.1	3.2.X	
SS.1(3)	SDD	10.1.5.2.1	3.2.X	
SS.1(4)	SDD	10.1.5.2.1	3.2.X	
SS.2(1)	SDD	10.1.5.2.1	3.2.X	
SS.2(2)	SDD	10.1.5.2.1	3.2.X	
ST.3(2)	SDD	10.1.5.1.1	3.1.1	
ST.3(5)	SDD	10.1.5.1.1	3.1.1	
	SDD	10.1.5.2.1	3.2.X	
ST.3(6)a	SDD	10.1.5.1.1	3.1.1	
	SDD	10.1.5.2.1	3.2.X	

Worksheet 2 - CSCI Level
Preliminary Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
Where is interoperating system info for next block of questions?				
SY.1(1)	SDD	10.1.5.2.1	3.2.X	
SY.1(2)	SDD	10.1.5.2.1	3.2.X	
SY.1(3)	SDD	10.1.5.2.1	3.2.X	
SY.1(4)	SDD	10.1.5.2.1	3.2.X	
SY.2(1)	SDD	10.1.5.2.1	3.2.X	
SY.2(2)	SDD	10.1.5.2.1	3.2.X	
SY.2(3)	SDD	10.1.5.2.1	3.2..X	
SY.4(1)	SDD	10.1.5.2.1	3.2.X	
SY.4(2)	SDD	10.1.5.2.1	3.2.X	
SY.4(3)	CRISD	10.1.5.1.1	3.1.1	Wrong phase (detailed)
TC.1(1)	SDD	10.1.5.2.1	3.2.X	Table not explicitly mentioned
VR.1(1)				Same as OP.1(16)

Worksheet 3A - CSCI Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
AM.1(3)				References 3B
AM.2(2)	SDD	10.1.6.1.2.2	4.X.Y.2	
AM.2(3)	SDD	10.1.6.1.2.2	4.X.Y.2	
AM.2(4)	SDD	10.1.6.1.2.2	4.X.Y.2	
AM.2(5)	SDD	10.1.6.1.2.2	4.X.Y.2	
AM.2(6)	SDD	10.1.6.1.2.2	4.X.Y.2	
AM.2(7)				References 3B
AM.3(2)	SDD	10.1.6.1.2.2	4.X.Y.2	
AM.3(3)	SDD	10.1.6.1.2.2	4.X.Y.2	
AM.3(4)	SDD	10.1.6.1.2.2	4.X.Y.2	
AP.1(1)				References 3B
AP.2(1)				References 3B
AP.2(2)				References 3B
AP.2(3)				References 3B
AP.2(4)				References 3B
AP.3(1)				References 3B
AP.4(1)				References 3B
AT.1(1)				References 3B
AT.1(2)a				Resource allocations are documented in the
AT.1(2)b				SDD (preliminary design)
AT.1(3)a				See above
AT.1(3)b				
AT.2(1)				References 3B
AT.2(2)				References 3B

Worksheet 3A - CSCI Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
AT.2(3)a				See AT.1(2) for this group
AT.2(3)b				
AT.3(1)a	IDD	10.1.5.2.1		
AT.3(1)b	IDD	10.1.5.2.4.1.1		
AT.3(2)a	IDD	10.1.5.2.1		
AT.3(2)b	IDD	10.1.5.2.4.1.1		
AU.1(2)				No basis suggested for estimating lines of code
AU.1(3)a	SDD	10.1.6.1	4.X	
AU.1(3)b	SDD	10.1.6.1.2.1	4.2.X	
AU.1(4)a				CSCI execution time estimates use information documented in the SDD (preliminary design). See Worksheet 2, these questions.
AU.1(4)b				
CL.1(7)a	SDD	10.1.5.1	3.1	
	SDD	10.1.6.1	4.X	
CL.1(8)a	SDD	10.1.5.1	3.1	
	SDD	10.1.6.1	4.X	
CL.2(1)	SDP	10.2.6.2.4	4.2.4	
	SDD	10.1.6.1.2.2	4.X.Y.2	Note: local data
	SDD	10.1.7	5	global data
	SDD	10.1.8.2	6.2	data file
CL.2(2)	SDD	10.1.6.1.2.2	4.X.Y.2	
CP.1(1)				References 3B
CP.1(2)				References 3B

Worksheet 3A - CSCI Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
CP.1(3)a	SDD	10.1.6.1.2.2	4.X.Y.2	Note: local data
	SDD	10.1.7	5	global data
	SDD	10.1.8.2	6.2	data file
CP.1(3)b	SDD	10.1.6.1.2.2	4.X.Y.2	
CP.1(4)				References 3B
CP.1(9)				References 3B
CP.1(10)				References 3B
CP.1(11)				Not a deliverable item
CS.1(1)				References 3B
CS.1(2)				References 3B
CS.1(3)				References 3B
CS.1(4)				References 3B
CS.1(5)				References 3B
CS.2(1)				References 3B
CS.2(2)				References 3B
CS.2(3)				References 3B
CS.2(6)				References 3B
DI.1(1)	SDD	10.1.5.1.1	3.1.1	
	SDD	10.1.6.1	4.X	
DO.1(1)	SDP	10.2.5.9	3.9	
DO.2(1)	SDD	10.1.6.1.2.2	4.X.Y.2	Subjective
DO.2(2)	SDD	10.1.6.1	4.X	*Graphic presentation not explicitly required*

Worksheet 3A - CSCI Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
DO.2(3)	All	10.1.2		
DO.2(4)	All	10.3.3	1.3	
DO.2(7)	STD	10.1.6.1.2.2	4.x.y.2	
EP.1(2)				References 3B
EP.1(3)	SDD	10.1.6.1.2.1		
EP.1(4)				References 3B
EP.1(5)	SDD	10.1.6.1.2.1	4.X.Y.1	
EP.1(6)				References 3B
EP.2(2)	SDD	10.1.6.1.2.1		Subjective; no prior criteria established
EP.2(4)				References 3B
EP.2(5)				References 3B
EP.2(6)	SDD	10.1.6.1.2.1	4.X.Y.1	Subjective; no prior criteria established
EP.2(7)				References 3B
ES.1(3)a	SDD	10.1.7	5	
ES.1(3)b	SDD	10.1.6.1.2.2	4.X.Y.2	
ES.1(4)	SDD	10.1.5.1.2	3.1.2	(Weak) Vague question
	SDD	10.1.5.1.3	3.1.3	
ES.1(6)				References 3B
ES.1(8)	SDD	10.1.8.2		
FS.1(1)				References 3B

Worksheet 3A - CSCI Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
GE.1(1)a	SDD	10.1.6.1	4.X	
GE.1(1)b	SDD	10.1.6.1.2.2	4.X.Y.2	
GE.2(1)a	SDD	10.1.6.1	4.X	
GE.2(1)b	SDD	10.1.6.1.2.2	4.X.Y.2	
GE.2(2)				References 3B
GE.2(3)				References 3B
GE.2(4)				References 3B
ID.1(1)				References 3B
ID. 1(3)				References 3B
ID.2(2)a	SDD	10.1.6.1	4.X	
ID.2(2)b	SDD	10.1.6.1.2.2	4.X.Y.2	
ID.2(3)a	SDD	10.16.1	4.X	
ID.2(3)b	SDD	10.1.6.1.2.2	4.X.Y.2	
ID.2(4)a	SDD	10.1.6.1	4.X	
ID.2(4)b	SDD	10.1.6.1.2.2	4.X.Y.2	
MO.1(2)	SDP	10.2.6.2.1	4.2.1	
MO.1(3)				References 3B
MO.1(4)				References 3B
MO.1(5)				References 3B
MO.1(6)				References 3B
MO.1(7)				References 3B
MO.1(8)				References 3B
MO.1(9)				References 3B

Worksheet 3A - CSCI Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
MO2(2)a	SDD	10.1.6.1	4.X	
MO.2(2)b	SDD	10.1.6.1.2.2	4.X.Y.2	
MO.2(3)a	SDD	10.1.6.1	4.X	
MO.2(3)b	SDD	10.1.6.1.2.2	4.X.Y.2	
MO.2(5)				References 3B
OP.1(2)	SDD	10.1.6.1.2.1	4.X.Y.1	
OP.1(3)	SDD	10.1.6.1.2.1	4.X.Y.1	
OP.1(10)a	SDD	10.1.6.1.2.2	4.X.Y.2	
OP.1(10)b	SDD	10.1.6.1.2.2	4.X.Y.2	
SD.3(5)				References 3B
SI.1(1)				Wrong phase (preliminary)
SI.1(2)				References 3B
SI.1(3)				References 3B
SI.1(4)				References 3B
SI.1(5)				References 3B
S1.1(6)	SDD	10.1.7	5	
S1.1(7)	SDD	10.1.7	5	
SI.1(10)	SDD	10.1.6.1	4.X	Hardware not specified
S1.3(1)				References 3B
S1.4(1)				References 3B

Worksheet 3A - CSCI Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
S1.4(2)				References 3B
S1.4(3)				References 3B
S1.4(4)				References 3B
S1.4(5)				References 3B
S1.4(14)	SDD	10.1.6.1.2.2	4.X.Y.2	Note: source code not yet available
SI.5(2)				References 3B
S1.5(2)				References 3B
S1.5(3)				References 3B
S1.6(1)				References 3B
ST.1(1)				References 3B
ST.1(2)				References 3B
ST.1(3)				References 3B
ST.1(4)				References 3B
ST.1(5)				References 3B
ST.1(6)a	SDD	10.1.6.1	4.X	
ST.1(6)b	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.2(1)				References 3B
ST.2(2)				References 3B
ST.2(3)				References 3B
ST.2(4)				References 3B
ST.3(2)	SDD	10.1.5.1.1 10.1.6.1.2.2	3.1.1 4.X.Y.Z	
ST.3(3)				References 3B
ST.3(4)	SDD	10.1.6.1.2.2	4.X.Y.2	

Worksheet 3A - CSCI Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
ST.4(1)				References 3B
ST.4(2)				References 3B
ST.4(3)a	SDD	10.1.7	5	
ST.4(3)b	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.4(4)				References 3B
ST.4(5)				References 3B
ST.5(1)				References 3B
ST.5(2)				References 3B
ST.5(3)				References 3B
ST.5(4)				References 3B
TC.1(1)	SDD	10.1.6.1.2.1	4.X.Y.1	
TC.1(2)	SDD	10.1.5.2.1	3.2.1	Wrong phase (preliminary)
TN.1(1)				Inappropriate level (SSS)
TN.1(2)				See above
TN 1(3)				See above
TN.1(4)				See above
VS.1(1)				References 3B
VS.1(2)				References 3B
VS.2(1)a	SDD	10.1.6.1	4.X	
VS.2(1)b	STP	10.1.6.1.3	4.X.3	
VS.3(1)	STP	10.1.6.1.2	4.X.2	

Worksheet 3A - CSCI Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
VS.3(2)	STP	10.1.6.1.	4.X	
VS.3(3)	STD	10.1.6.1.1.3	4.X.Y3	
	STD	10.1. 6.1.1.4	4.X.Y4	

Worksheet 3B - Unit Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
AM.1(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
AM.2(7)d	SDD	10.1.6.1.2.2	4.X.Y.2	
AP.1(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
AP.2(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
AP.2(1)e	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
AP.2(2)d	SDP	10.2.6.2.4	4.2.4	
AP.2(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
AP.2(4)d	SDD	10.1.6.1.2.2	4.X.Y.2	
AP.3(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
AP.4(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
AT.1(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
	SDD	10.1.8.2	6.X	
AT.2(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
AT.2(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	Subjective
CP.1(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	Subjective

Worksheet 3B - Unit Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
CP.1(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	
CP.1(2)e	SDD	10.1.6.1.2.2	4.X.Y.2	
CP.1(4)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
CP.1(4)e	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
CP.1(9)d	SDD	10.1.6.1.2.2	4.X.Y.2	
CP.1(10)d	SDD	10.1.6.1.2.2	4.X.Y.2	
CS.1(1)d	SDP	10.2.6.2.3	4.2.3	
CS.1(2)d	SDP	10.2.6.2.3	4.2.3	
	IDD	10.1.5.2.4.1		
CS.1(3)d	SDP	10.2.6.2.3	4.2.3	
	IDD	10.1.5.2		
CS.1(4)d	SDD	10.1.6.1.2.2	4.X.Y.2	
CS.1(5)d				Scope of term "references" unknown
CS.2(1)d	SDP	10.2.6.2.3	4.2.3	
	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7		

Worksheet 3B - Unit Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
CS.2(2)d	SDP	10.2.6.2.3	4.2.3	
	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7		
CS.2(3)d	SDP	10.2.6.2.3	4.2.3	
	SDD	10.1.7		
CS.2(6)d				(same as CS.1(5)d)
EP.1(2)d				Need source code (SPS)
EP.1(2)e				Need source code (SPS)
EP.1(4)d				Need source code (SPS)
EP.1(4)e				Need source code (SPS)
EP.1(6)d				Need source code (SPS)
EP.1(6)e				Need source code (SPS)
EP.2(4)d				Need source code (SPS)
EP.2(4)e				Need source code (SPS)
EP.2(5)d				Need source code (SPS)
EP.2(5)e				Need source code (SPS)
EP.2(7)d				Need source code (SPS)
EP.2(7)e				Need source code (SPS)
ES.1(6)d	SDD	10.1.6.1.2.2	4.X.Y.2	

Worksheet 3B - Unit Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
FS.1(1)d	SDD	10.1.6.1.2.1	4.X.Y.1	
GE.2(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	
GE.2(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
GE.2(4)d	SDD	10.1.6.1.2.2	4.X.Y.2	
ID.1(1)d				No method specified for estimating lines of code
ID.1(1)e	SDD	10.1.6.1.2.2	4.X.Y.2	
ID.1(3)d	SDP SPS	10.2.6.2.4 10.1.5.2	4.2.4 3.2	
MO.1(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
MO.1(4)d	SDD	10.1.6.1.2.2	4.X.Y.2.	
MO.1(4)e	SDD	10.1.6.1.2.2	4.X.Y.2	
MO.1(5)d	SDD	10.1.6.1.2.2	4.X.Y.2.	
MO.1(6)d	SDD	10.1.6.1.2.2	4.X.Y.2	
MO.1(7)d	SDD	10.1.6.1.2.2	4.X.Y.2	
MO.1(8)d	SDD	10.1.6.1.2.2	4.X.Y.2	
MO.1(9)d	SDD	10.1.6.1.2.1	4.X.Y.1	

Worksheet 3B - Unit Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
MO.2(5)d	SDD	10.1.6.1.2.2	4.X.Y.2	
SD.3(5)d	SDD	10.1.6.1.2.1	4.X.Y.1	
SI.1(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	
SI.1(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
SI.1(4)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDP	10.2.6.2.4	4.2.4	
SI.1(5)d	SDD	10.1.6.1.2.2	4.X.Y.2	
SI.1(5)e	SDD	10.1.6.1.2.2	4.X.Y.2	
SI.3(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)
SI.3(1)e	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)
SI.4(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
SI.4(2)d				No method specified for estimating lines of code
SI.4(2)e				Need source code (SPS)
SI.4(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)
SI.4(3)e	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)
SI.4(4) d	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)
SI.4(4) e	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)
SI.4(5)d				Need source code (SPS)

Worksheet 3B - Unit Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
SI.5(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
SI.5(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	
SI.5(2)e	SDD	10.1.6.1.2.2	4.X.Y.2	
SI.5(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
SI.6(1)d				Need source code (SPS)
SI.6(1)e				Need source code (SPS)
ST.1(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.1(2) d				Need source code (SPS)
ST.1(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.1(3)e	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.1(4)d	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.1(5)d	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.2(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)
ST.2(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)
ST.2(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.2(4)d	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)

Worksheet 3B - Unit Level
Detailed Design

Metric Element	DID	DID Paragraph	Document Reference	Comments
ST.3(3 d	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.4(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
ST.4(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
ST.4(4)d	SDD	10.1.6.1.2.2		
ST.4(5)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
ST.5(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
ST.5(2)d				Need source code (SPS)
ST.5(3)d				Need source code (SPS)
ST.5(4)d				Need source code (SPS)
VS.1(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	Need PDL/flowchart; otherwise need source code (SPS)
VS.1(1)e	STP	10.1.6.1.4.1	4.X.4.Y	
VS.1(2) d	SDD	10.1.6.1.2.2	4.X.Y.2	
VS.1(2) e	STP	10.1.6.1.4.1	4.X.4.Y	

Worksheet 4A - CSCI Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
AC.1(8)	STR	10.1.6.1.1.1	4.X.Y.1	Test Results, w/respect to Level 1 accuracy req'ts
AM.1(3)				References 4B
AM.2(2)	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
AM.2(3)	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
AM.2(4)	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
AM.2(5)	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
AM.2(6)	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
AM.2(7)				References 4B
AM.3(2)	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
AM.3(3)	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
AM.3(4)	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
AP.1(1)				References 4B
AP.2(1)				References 4B
AP.2(2)				References 4B
AP.2(3)				References 4B

Worksheet 4A - CSCI Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
CL.1(7)a	SDD	10.1.6.1	4.X	Count units in design
	SPS	10.1.5.2	3.2	Count units in code
CL.1(8)a	SDD	10.1.6.1	4.X	Count units in design
	SPS	10.1.5.2	3.2	Count units in code
CL.2(1)	SDP	10.2.6.2.4	4.2.4	
	SPS	10.1.5.2	3.2	
CL.2(2)a	SDD	10.1.6.1.2.2	4.X	Count units in design
	SPS	10.1.5.2	3.2	Count units in code
CP.1(2)				References 4B
CP.1(3)a	SDD	10.1.6.1.2.2	4.X.Y.2	Note: local data
	SDD	10.1.7	5	global data
	SDD	10.1.8.2	6.2	data file
	SPS	10.1.5.2	3.2	Actual unit data listings (ie. definitions)
CP.1(3)b	STR	10.1.6.1.1.1	4.X.Y.1	Test Results for data item usage
	SPS	10.1.5.2	3.2	CSCI source code listings
CP.1(4)				References 4B
CP.1(9)				References 4B
CP.1(10)				References 4B
CP.1(11)				Not a deliverable item
CS.1(2)				References 4B
CS.1(3)				References 4B
CS.1(4)				References 4B

Worksheet 4A - CSCI Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
CS.1(5)				References 4B
CS.2(1)				References 4B
CS.2(2)				References 4B
CS.2(3)				References 4B
CS.2(6)				References 4B
DO.1(1)	SDP	10.2.5.9	3.9	
DO.2(1)	SDD	10.1.6.1.2.2	4.X.Y.2	Subjective
	SPS	10.1.5.2	3.2	
DO.2(3)	All	10.1.2		
DO.2(4)	All	10.3.3	1.3	
EP.1(2)				References 4B
EP.1(3)				References 4B
EP.1(4)				References 4B
EP.1(5)	SPS	10.1.5.4		Measured resource utilization
	SDD	10.1.6.1.2.1	4.X.Y.1	Design of units may require overlay usage
EP.1(6)				References 4B
EP.2(4)				References 4B
EP.2(5)				References 4B
EP.2(7)				References 4B
ES.1(3)a	SPS	10.1.5.2	3.2	
ES.1(3)b	SDD	10.1.6.1.2.2	4.X.Y.2	Design considerations on aliases, to help
	SPS	10.1.5.2	3.2	
ES.1(4)	SPS	10.1.5.4		

Worksheet 4A - CSCI Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
ES.1(6)				References 4B
ES.1(7)a	SPS	10.1.5.2	3.2	
ES.1(7)b	SDD	10.1.6.1.2.2	4.X.Y.2	Special compiler usage mentioned in design
	SPS	10.1.5.3	3.3	Compiler/Assembler usage
ES.1(8)				
FS.1(1)				References 4B
GE.1(1)a	SPS	10.1.5.2	3.2	
GE.1(1)b	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
GE.2(1)a	SPS	10.1.5.2	3.2	
GE.2(1)b	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
GE.2(2)				References 4B
GE.2(3)				References 4B
GE.2(4)				References 4B
ID.1(1)				References 4B
ID. 1(3)				References 4B
ID.2(2)a	SPS	10.1.5.2	3.2	
ID.2(2)b	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
ID.2(3)a	SPS	10.1.5.2	3.2	
ID.2(3)b	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
ID.2(4)a	SPS	10.1.5.2	3.2	

Worksheet 4A - CSCI Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
ID.2(4)b	SPS	10.1.5.2	3.2	
MO.1(2)	SDP	10.2.6.2.1	4.2.1	
	STP	10.1.6.1.1		
MO.1(3)				References 4B
MO.1(4)				References 4B
MO.1(5)				References 4B
MO.1(6)				References 4B
MO.1(7)				References 4B
MO.1(8)				References 4B
MO.1(9)				References 4B
M0.2(2)a	SDD	10.1.6.1	4.X	In design
	SPS	10.1.5.2	3.2	In actual code
MO.2(2)b	SDD	10.1.6.1.2.2	4.X.Y.2	In design
	SPS	10.1.5.2	3.2	In actual code
MO.2(3)a	SDD	10.1.6.1	4.X	In design
	SPS	10.1.5.2	3.2	
MO.2(3)b	SDD	10.1.6.1.2.2	4.X.Y.2	In design
	SPS	10.1.5.2	3.2	In actual code
M0.2(5)				References 4B
OP.1(2)	STR	10.1.7.1		
OP.1(3)	STR	10.1.6.1.1.1	4.X.Y.1	Test Results for user interface
OP.1(10)a	SPS	10.1.5.2	3.2	
OP.1(10)b	SPS	10.1.5.2	3.2	
SD.3(5)				References 4B

Worksheet 4A - CSCI Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
SI.1(2)				References 4B
SI.1(3)				References 4B
SI.1(4)				References 4B
SI.1(5)				References 4B
S1.1(6)	SPS	10.1.5.2	3.2	
S1.1(7)	SPS	10.1.5.2	3.2	
SI.1(10)	SDD	10.1.6.1	4.X	Hardware not specified
	SPS	10.1.5.2	3.2	
S1.3(1)				References 4B
S1.4(1)				References 4B
S1.4(2)				References 4B
S1.4(3)				References 4B
S1.4(4)				References 4B
S1.4(5)				References 4B
S1.4(14)	SPS	10.1.5.2	3.2	
	STR	10.1.7.1		CSCI capabilities per test results
SI.5(1)				References 4B
S1.5(2)				References 4B
S1.5(3)				References 4B
S1.6(1)				References 4B
ST.1(1)				References 4B
ST.1(2)				References 4B
ST.1(3)				References 4B
ST.1(4)				References 4B
ST.1(5)				References 4B
ST.1(6)a	SPS	10.1.5.2	3.2	
ST.1(6)b	SDD	10.1.6.1.2.2	4.X.Y.2	In design

Worksheet 4A - CSCI Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
	SPS	10.1.5.2	3.2	In actual code
ST.2(1)				References 4B
ST.2(2)				References 4B
ST.2(3)				References 4B
ST.2(4)				References 4B
ST.3(3)				References 4B
ST.3(4)b	SDD	10.1.6.1.2.2	4.X.Y.2	In design
	SPS	10.1.5.2	3.2	In actual code
ST.4(1)				References 4B
ST.4(2)				References 4B
ST.4(3)a	SPS	10.1.5.2	3.2	
ST.4(3)b	SDD	10.1.6.1.2.2	4.X.Y.2	In design
	SPS	10.1.5.2	3.2	In actual code
ST.4(4)				References 4B
ST.4(5)				References 4B
ST.5(1)				References 4B
ST.5(2)				References 4B
ST.5(3)				References 4B
ST.5(4)				References 4B
VS.1(1)				References 4B
VS.1(2)				References 4B

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
AM.1(3)d	STR	10.1.6.1.1.1	4.X.Y.1	
AM.2(7)d	STR	10.1.6.1.1.1	4.X.Y.1	
AP.1(1)d	SPS	10.1.5.2	3.2	
AP.2(1)d	SPS	10.1.5.2	3.2	
AP.2(1)e	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
	SPS	10.1.5.2	3.2	
AP.2(2)d	SPS	10.1.5.2	3.2	
AP.2(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
AP.2(4)d	SPS	10.1.5.2	3.2	
AP.3(1)d	SPS	10.1.5.2	3.2	
AP.3(2)d	SPS	10.1.5.2	3.2	
AP.3(2)e	SPS	10.1.5.2	3.2	
AP.4(1)d	SPS	10.1.5.2	3.2	
AT.1(1)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
	SPS	10.1.5.2	3.2	
AT.2(1)d	SPS	10.1.5.2	3.2	

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
AT.2(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	Subjective; possibly wrong phase (design)
CP.1(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	
CP.1(2)e	SPS	10.1.5.2	3.2	
CP.1(4)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
CP.1(4)e	SDD	10.1.6.1.2.2	4.X.Y.2	
	SDD	10.1.7	5	
	SPS	10.1.5.2	3.2	
CP.1(9)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
CP.1(10)d	SPS	10.1.5.2	3.2	
CS.1(1)d	SDP	10.2.6.2.3	4.2.3	
	STR	10.1.6.1.1.1	4.X.Y.1	
CS.1(2)d	SDP	10.2.6.2.3	4.2.3	
	STR	10.1.6.1.1.1	4.X.Y.1	
CS.1(3)d	SDP	10.2.6.2.3	4.2.3	
	STR	10.1.6.1.1.1	4.X.Y.1	
CS.1(4)d	SDP	10.2.6.2.3	4.2.3	
	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
CS.1(5)d	SPS	10.1.5.2	3.2	

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
CS.2(1)d	SDP	10.2.6.2.3	4.2.3	
	SPS	10.1.5.2	3.2	
CS.2(2)d	SDP	10.2.6.2.3	4.2.3	
	SPS	10.1.5.2	3.2	
CS.2(3)d	SDP	10.2.6.2.3	4.2.3	
	SPS	10.1.5.2	3.2	
CS.2(6)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
EP.1(2)d	SPS	10.1.5.2	3.2	
EP.1(2)e	SPS	10.1.5.2	3.2	
EP.1(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
EP.1(3)e	SDD	10.1.6.1.2.2	4.X.Y.2	
EP.1(4)d	SPS	10.1.5.2	3.2	
EP.1(4)e	SPS	10.1.5.2	3.2	
	STR	10.1.7.1	5.1	
EP.1(6)d	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
EP.1(6)e	SPS	10.1.5.2	3.2	
	STR	10.1.7.1	5.1	
EP.2(4)d	SPS	10.1.5.2	3.2	
EP.2(4)e	SPS	10.1.5.2	3.2	
EP.2(5)d	SPS	10.1.5.2	3.2	

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
EP.2(5)e	SPS	10.1.5.2	3.2	
EP.2(7)d	SPS	10.1.5.2	3.2	
EP.2(7)e	SPS	10.1.5.2	3.2	
ES.1(6)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
FS.1(1)d	SDD	10.1.6.1.2.1	4.X.Y.1	
	STR	10.1.7.1	5.1	
GE.2(2)d	SPS	10.1.5.2	3.2	
GE.2(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	STR	10.1.7.1	5.1	
GE.2(4)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	STR	10.1.7.1.	5.1	
ID.1(1)d	SPS	10.1.5.2	3.2	
ID.1(1)e	SPS	10.1.5.2	3.2	
ID.1(3)d	SPS	10.1.5.2	3.2	
MO.1(3)d	SPS	10.1.5.2	3.2	
MO.1(4)d	SPS	10.1.5.2	3.2	
MO.1(4)e	SPS	10.1.5.2	3.2	
MO.1(5)d	SDD	10.1.6.1.2.2	4.X.Y.2.	
	STR	10.1.6.1.1.1	4.X.Y.1	

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
	SPS	10.1.5.2	3.2	
MO.1(6)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
	SPS	10.1.5.2	3.2	
MO.1(7)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
MO.1(8)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
MO.1(9)d	SDD	10.1.6.1.2.1	4.X.Y.1	Wrong phase? Testing results? Subjective?
MO.2(5)d	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
SD.1(1)d	SPS	10.1.5.2	3.2	
SD.1(1)e	SPS	10.1.5.2	3.2	
SD.2(1)d	SPS	10.1.5.2	3.2	
SD.2(2)d	SPS	10.1.5.2	3.2	
SD.2(3)d	SPS	10.1.5.2	3.2	
SD.2(4)d	SPS	10.1.5.2	3.2	
SD.2(5)d	SPS	10.1.5.2	3.2	
SD.2(6)d	SPS	10.1.5.2	3.2	

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
SD.2(7)d	SPS	10.1.5.2	3.2	
SD.2(8)d	SPS	10.1.5.2	3.2	
SD.3(1)d	SDD	10.1.6.1.2.2	4.X.Y.1	
	SPS	10.1.5.2	3.2	
	SPS	10.1.5.3	3.3	
SD.3(2)d	SPS	10.1.5.2	3.2	
SD.3(3)d	SPS	10.1.5.2	3.2	
SD.3(4)d	SPS	10.1.5.2	3.2	
SD.3(4)e	SPS	10.1.5.2	3.2	
SD.3(5)d	SDP	10.1.6.2.4		
	SPS	10.1.5.2	3.2	
SD.3(6)d	STR	10.1.6.1.1.1	4.X.Y.1	
	SPS	10.1.5.2	3.2	
SI.1(2)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	STR	10.1.7.1	5.1	
SI.1(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	STR	10.1.7.1	5.1	
SI.1(4)d	SPS	10.1.5.2	3.2	
SI.1(5)d	SDD	10.1.6.1.2.2	4.X.Y.2	

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
SI.1(5)e	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
SI.3(1)d	SPS	10.1.5.2	3.2	
SI.3(1)e	SPS	10.1.5.2	3.2	
SI.4(1)d	SPS	10.1.5.2	3.2	
SI.4(2)d	SPS	10.1.5.2	3.2	
SI.4(2)e	SPS	10.1.5.2	3.2	
SI.4(3)d	SPS	10.1.5.2	3.2	
SI.4(3)e	STR	10.1.7.1	5.1	
SI.4(4)d	SPS	10.1.5.2	3.2	
SI.4(4)e	STR	10.1.7.1	5.1	
SI.4(5)d	SPS	10.1.5.2	3.2	
	STR	10.1.7.1	5.1	
SI.4(6)d	SPS	10.1.5.2	3.2	
SI.4(6)e	SPS	10.1.5.2	3.2	
SI.4(7)d	SPS	10.1.5.2	3.2	
SI.4(7)e	STR	10.1.6.1.1.1	4.X.Y.1	
SI.4(8)d	SPS	10.1.5.2	3.2	

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
SI.4(8)e	STR	10.1.6.1.1.1	4.X.Y.1	
SI.4(9)d	SPS	10.1.5.2	3.2	
SI.4(9)e	SPS	10.1.5.2	3.2	
SI.4(10)d	SPS	10.1.5.2	3.2	
SI.4(10)e	STR	10.1.6.1.1.1	4.X.Y.1	
SI.4(11)d	SPS	10.1.5.2	3.2	
SI.4(11)e	STR	10.1.6.1.1.1	4.X.Y.1	
SI.4(12)d	STR	10.1.6.1.1.1	4.X.Y.1	
SI.4(13)	SDP	10.2.6.2.4	4.2.4	
	SPS	10.1.5.2	3.2	
SI.5(1)d	SPS	10.1.5.2	3.2	
SI.5(2)d	SPS	10.1.5.2	3.2	
SI.5(2)e	STR	10.1.6.1.1.1	4.X.Y.1	
SI.5(3)d	SDD	10.1.6.1.2.1	4.X.Y.2	
	STR	10.1.7.1	5.1	
SI.6(1)d	SPS	10.1.5.2	3.2	
SI.6(1)e	SPS	10.1.5.2	3.2	
ST.1(1)d	SPS	10.1.5.2	3.2	
ST.1(2) d	SPS	10.1.5.2	3.2	

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
ST.1(3)d	SPS	10.1.5.2	3.2	
ST.1(3)e	STR	10.1.6.1.1.1	4.X.Y.1	
ST.1(4)d	SPS	10.1.5.2	3.2	
ST.1(5)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	SPS	10.1.5.2	3.2	
ST.2(1)d	SPS	10.1.5.2	3.2	
ST.2(2)d	SPS	10.1.5.2	3.2	
ST.2(3)d	SDD	10.1.6.1		
	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
ST.2(4)d	SPS	10.1.5.2	3.2	
ST.2(5)d	SPS	10.1.5.2	3.2	
ST.3(3)d	SDD	10.1.6.1.2.2	4.X.Y.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
ST.4(1)d	SPS	10.1.5.2	3.2	
ST.4(2)d	SPS	10.1.5.2	3.2	
ST.4(4)d	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
ST.4(5)d	SDD	10.1.6.1.2.2	4.X.Y.2	

Worksheet 4B - Unit Level
Code and Unit to CSCI Testing

Metric Element	DID	DID Paragraph	Document Reference	Comments
	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
ST.5(1)d	SPS	10.1.5.2	3.2	
	STR	10.1.7.1	5.1	
ST.5(2)d	SPS	10.1.5.2	3.2	
	STR	10.1.7.1	5.1	
ST.5(3)d	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
ST.5(4)d	SPS	10.1.5.2	3.2	
	STR	10.1.7.1	5.1	
VS.1(1)d	SPS	10.1.5.2	3.2	
VS.1(1)e	STP	10.1.6.1.4.1	4.X.4.Y	
	STR	10.1.6.1.1.1	4.X.Y.1	
VS.1(2) d	SPS	10.1.5.2	3.2	
	STR	10.1.6.1.1.1	4.X.Y.1	
VS.1(2) e	STP	10.1.6.1.4.1	4.X.4.Y	
	STR	10.1.6.1.1.1	4.X.Y.1	

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.